Technická univerzita v Košiciach

Fakulta elektrotechniky a informatiky

Scolopendra – Softvérové rozšírenie šesťnohej robotickej platformy

Diplomová práca

2023 **Bc. Samuel Titko**

Technická univerzita v Košiciach

Fakulta elektrotechniky a informatiky

Scolopendra – Softvérové rozšírenie šesťnohej robotickej platformy

Diplomová práca

Študijný program:	Inteligentné systémy
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra kybernetiky a umelej inteligencie (KKUI)
Školiteľ:	doc. Ing. Dr. Ján Vaščák

Košice 2023 **Bc. Samuel Titko**

Abstrakt v SJ

Cieľom tejto diplomovej práce je rozšíriť softvér hexapodického robota Scolopendra navrhnutého v mojej bakalárskej práci. Diplomová práca sa zameriava na vytvorenie virtuálnych modelov vo vizualizácii a simulácii, grafického užívateľského rozhrania pre riadenie robota, softvérového balíka z celého projektu a architektúr prepájateľných aplikácií. Práca je rozdelená na teoretickú analýzu, popis aplikovaných technológií a praktickú implementáciu všetkých technológií do vlastného softvéru. V závere sú zhodnotené dosiahnuté ciele celej práce.

Kľúčové slová

grafické užívateľské rozhranie, komunikácia, simulácia, softvérový balík, virtuálny model, vizualizácia, Webots

Abstrakt v AJ

The aim of this diploma thesis is to extend software of hexapodic robot Scolopendra designed in my bachelor thesis. The diploma thesis is concentrating to create virtual models in visualization and simulation, graphical user interface for robot control, software package from whole project and architectures of connectible applications. Thesis is divided into theoretical analysis, description of applied technologies and practical implementation all of these technologies into custom software. In conclusion, the accomplished results of the entire thesis are evaluated.

Kľúčové slová v AJ

graphical user interface, communication, simulation, software package, virtual model, visualization, Webots

68241

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra kybernetiky a umelej inteligencie

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**
Študijný program: **Inteligentné systémy**

Názov práce:

Scolopendra – Softvérové rozšírenie šesťnohej robotickej platformy

Scolopendra – Software Extension of a Six-legged Robotic Platform

Študent: **Bc. Samuel Titko**
Školiteľ: **doc. Dr. Ing. Ján Vaščák**
Školiace pracovisko: **Katedra kybernetiky a umelej inteligencie**
Konzultant práce:
Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:

1. Popísať technológie pre tvorbu virtuálnych modelov mobilných robotov.
2. Analyzovať potreby grafického užívateľského rozhrania pre riadenie robota Scolopendra.
3. Navrhnúť štruktúru softvérového balíka pre prácu so zvoleným robotom.
4. Vytvoriť knižnicu a programovú podporu pre prácu s fyzickým a virtuálnym modelom robota.
5. Experimentálne otestovať a vyhodnotiť funkčnosť softvérového návrhu.
6. Vypracovať dokumentáciu podľa pokynov vedúceho záverečnej práce.

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 21.04.2023
Dátum zadania diplomovej práce: 31.10.2022

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY -12-

.....
prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice 21. 4. 2023 .....

Vlastnoručný podpis

Poďakovanie

Chcel by som sa poďakovať svojmu školiteľovi doc. Dr. Ing. Jánovi Vaščákovi za odbornú asistenciu a pomoc pri tvorbe a písaní tejto diplomovej práce, celej svojej rodine za podporu v tom, čo ma baví, svojej priateľke Sofii za jej podporu a trpezlivosť počas môjho štúdia na TUKE, spolužiakovi a dlhoročnému priateľovi Oliverovi Kudziovi za vzájomnú pomoc počas celého štúdia a bývalému učiteľovi zo strednej školy Ing. Jánovi Motešickému za jeho motiváciu, ktorá ma uviedla do oblasti robotiky a programovania.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam symbolov a skratiek

API	Application Program Interface
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CLI	Command Line Interface
CSV	Comma Separated Values
D3D	Direct3D
FoV	Field Of View
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTML	HyperText Markup Language
HMI	Human Machine Interface
IPC	Inter-process Communication
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
JSON	JavaScript Object Notation
LIFO	Last In, First Out
ODE	Open Dynamics Engine
PWM	Pulse Width Modulation
STL	Stereolitography
TCP	Transmission Control Protocol

TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
UI	User Interface
UML	Unified Modeling Language
UX	User eXperience
Wi-Fi	Wireless Fidelity
WPF	Windows Presentation Foundation
XAML	eXtensible Application Markup Language
XML	eXtensible Markup Language

Úvod

Bakalárska práca s názvom *Scolopendra – Návrh riadenia šesťnohej robotickej platformy* bola úvodom do oblasti kráčajúcich robotov so šiestimi končatinami. Cieľom práce bola teoretická analýza šesťnohých robotov – hexapodov, ich využitie v praxi, návrh dizajnu konštrukcie vlastného modelu, výroba konštrukcie navrhnutého modelu prostredníctvom 3D tlače, zapojenie elektronických komponentov, a matematický rozbor kinematického modelu robota a pohybov končatín. Práca bola viac orientovaná na dizajn konštrukcie, návrh hardvéru robota a matematické výpočty s minimálnym dôrazom na vývoj zložitejšieho riadiaceho softvéru.

V súčasnosti neexistuje možnosť vyvíjania a testovania softvéru robota Scolopendra bez fyzickej kópie navrhnutého modelu. Vytvorenie nového modelu je finančne a časovo náročné. Vývoj komplexnejšieho softvéru pre navrhnutý prototyp je z programátorského hľadiska obtiažny, pretože programátor musí vedieť aj obsluhovať a servisovať fyzický hardvér. Pôvodný softvér robota je funkčný, ale neoptimálny, z dôvodu veľkého množstva práce na hardvérovej realizácii. Štruktúra softvéru je nevhodná pre ďalšie programátorské účely. Zabezpečenie ovládania robota v bakalárskej práci je riešené užívateľskou konzolovou aplikáciou. Jej obsluha je náročná a neintuitívna pre neskúseného užívateľa a umožňuje len spúšťanie predvypočítaných sekvencií pohybov priamo na fyzickom modeli robota. Slúži iba na demonštračné účely pohybov a obsluha taktiež vyžaduje od užívateľa aj určité programátorské znalosti.

Z týchto dôvodov nadväzuje diplomová práca na bakalársku prácu. Zameriava sa na softvérové rozšírenie robota Scolopendra. Softvérové rozšírenie bude zamerané na vytvorenie usporiadanej štruktúry adresárov a súborov, ktoré majú vzájomný logický súvis a budú tak vytvárať vhodné prostredie pre vývoj aplikácií. Ďalej bude rozšírená o virtuálne modely v prostredí vizualizácie a simulácie. Virtuálne modely uľahčia prácu s modelom robota každému programátorovi, ktorý je neskúsený v oblasti elektroniky, 3D tlače, obsluhy robota a jeho servisu. Vývoj a testovanie softvéru

tak bude možný aj bez jeho aplikovania na fyzický model. Pri vytváraní komplexného softvéru zloženého z viacerých aplikácií, ktoré musia navzájom komunikovať, bude potrebné implementovať štandardné komunikačné rozhranie. Posledným rozšírením softvéru bude vytvorenie aplikácie užívateľského rozhrania pre ovládanie ľubovoľného modelu robota, ktorého obsluha bude intuitívna a prijateľná aj pre ľudí bez programátorských znalostí.

Práca sa nezameriava na oblasť vývoja nového hardvéru ani nijako nerozširuje dizajn konštrukcie alebo elektroniku navrhnutého hexapoda Scolopendra z bakalárskej práce.

1 Formulácia úlohy

Zadanie diplomovej práce spočíva vo vytvorení virtuálnych modelov robota vo vizualizácii a simulácii, vytvorení knižnice pre prácu so softvérovými balíkmi, vytvorení softvérového balíka špecializovaného pre programovanie hexapoda Scolopendra z mojej bakalárskej práce, vytvorenie grafického užívateľského rozhrania a vytvorení a experimentálnom otestovaní architektúr prepájateľných aplikácií.
Riešenie týchto úloh si vyžaduje vykonať nasledujúce kroky:

- **Popísať technológie pre tvorbu virtuálnych modelov mobilných robotov**

  Úlohou tohto bodu je vytvoriť teoretický prehľad technológií a nástrojov podporujúcich tvorbu virtuálnych 3D modelov robotov, ich vzájomné porovnanie a výhody ich používania. Prostredníctvom zvolených nástrojov budú vytvorené virtuálne kópie fyzického modelu robota Scolopendra.

- **Analyzovať potreby grafického užívateľského rozhrania pre riadenia robota Scolopendra**

  Ďalším krokom je analýza potreby vytvorenia aplikácie grafického užívateľského rozhrania. Prostredníctvom tohto rozhrania bude môcť aj užívateľ bez programátorských znalostí ovládať všetky vytvorené modely robota Scolopendra (virtuálne aj fyzický).

- **Vytvoriť knižnicu a programovú podporu pre prácu s fyzickým a virtuálnym modelom robota**

  Vytvorenie knižnice pre prácu s virtuálnym a fyzickým modelom robota zahŕňa:

  - transformáciu časti zdrojových kódov z bakalárskej práce do podoby knižnice s modulmi a ich optimalizáciu z výpočtového hľadiska,
  - pridanie nových modulov podporujúcich tvorbu vizualizácie,

  - vytvorenie modulu využívajúceho štandardné komunikačné rozhranie, určené pre komunikáciu medzi procesmi spustených aplikácií,
  - vytvorenie modulu pre prácu so softvérovými balíkmi a ďalších dodatočných modulov so všeobecným využitím.

- **Navrhnúť štruktúru softvérového balíka pre prácu so zvoleným robotom**

  Táto úloha sa zameriava na vytvorenie softvérového balíka s obsahom špeciálne prispôsobeným na vývoj aplikácií pre modely robotov Scolopendra. Balík bude rozširovať základnú funkcionalitu knižnice o špecifické funkcie a konfigurácie. Obsah balíka tak bude prispôsobený na rýchly vývoj špecializovaných aplikácií pre hexapoda a zároveň bude obsahovať aplikáciu grafického rozhrania pre jeho riadenie.

- **Experimentálne otestovať a vyhodnotiť funkčnosť softvérového návrhu**

  Posledným krokom hlavnej časti bude otestovať funkčnosť všetkých možných kombinácií prepojení medzi aplikáciou užívateľského rozhrania a vytvorenými riadiacimi aplikáciami modelov robota Scolopendra z modulov softvérového balíka. V procese testovania bude overovaná funkčnosť a riešenia V rámci experimentu bude overovaná výpočtová náročnosť aplikácií na testovací hardvér.

- **Vypracovať dokumentáciu**

  Súčasťou práce budú dve príručky – systémová a používateľská. Systémová príručka bude obsahovať dokumentáciu zdrojových kódov knižnice a softvérového balíka. Používateľská príručka bude popisovať inštaláciu softvéru a obsluhu grafického užívateľského rozhrania v spojení s riadiacimi aplikáciami modelov.

2 Teoretická analýza

Softvérové rozšírenie robota Scolopendra zasahuje do viacerých oblastí vývoja softvéru pre roboty, ktorých teoretickým rozborom sa zaoberá táto kapitola.

2.1 Virtuálny model

Ručne vytvárané technické nákresy a technické špecifikácie produktov v súčasnosti nahradili virtuálne modely. Virtuálny model je digitálnou kópiou, niekedy označovanou termínom digitálne dvojča z angl. *digital twin* a zároveň virtuálnou reprezentáciou fyzického systému alebo procesu – fyzický model v reálnom svete. Virtuálne modely sú postavené na počítačom podporovanom dizajne (CAD) a počítačom podporovanom inžinierstve (CAE). CAD systémy umožňujú vytvárať modely prostredníctvom 2D alebo 3D grafiky. CAE systémy slúžia na vytváranie simulačných modelov. V praxi slúži pre vizualizačné, simulačné, integračné, testovacie a monitorovacie účely. Oblasť mobilnej robotiky taktiež využíva myšlienku virtuálneho modelu pre navrhovaného robota primárne na vizualizačné, simulačné a testovacie účely[1].

2.1.1 Počítačová vizualizácia

Počítačová vizualizácia je metóda vytvárania 2D alebo 3D zobrazení v podobe grafov, diagramov, schém alebo modelov z akumulovaných dát prostredníctvom vizualizačného softvéru. Využíva zmyslové reprezentácie vizuálneho charakteru, ktorých úlohou je pochopenie abstraktných údajov, budovanie hypotéz a uvažovanie nad ich podstatou. Neinteraktívne vizualizácie sú najbežnejším typom, ktoré jednorázovo zobrazujú dáta bez ich zmeny v čase alebo možnosti zásahu užívateľom do zobrazovacieho procesu. Existujú však aj dynamické – interaktívne vizualizácie. Interaktivita pre užívateľa vizualizačného softvéru môže byť zabezpečená animáciami alebo možnosťou ovládania určitých funkcií vizualizácie vstupnými perifériami počítača. Možnosť interaktivity však prináša dva navzájom súvisiace problémy. V dneš-

nej dobe má vizualizácia čoraz väčšie uplatnenie v širokom spektre výskumných a vývojových oblastí napríklad:

- fyzike,
- chémii,
- medicíne,
- strojárstve,
- priemysle,
- dátovej analytike,
- a mnoho ďalších.

Okrem vymenovaných oblastí sa vizualizácia uplatňuje aj v robotike v neinteraktívnej a interaktívnej forme. Tu sa využíva na 2D alebo 3D zobrazenie stavu virtuálneho modelu robota a rôznych iných objektov, ktoré v reálnom svete nie je možné nijako reprezentovať napr. trajektórie pohybu objektov, koordinačné rámce alebo prechodné stavy systému.[2] Neinteraktívnu vizualizáciu je možné využiť na zobrazenie samotného modelu robota. Interaktívnou vizualizáciou je možné meniť pohľad na virtuálny model robota, meniť jeho stav v čase prostredníctvom animácie alebo reagovať určitým spôsobom na užívateľský vstup[3].

2.1.2 Počítačová simulácia

Počítačová simulácia je snaha modelovať skutočné alebo hypotetické situácie prostredníctvom výpočtovej techniky, tak, aby bolo možné študovať fungovanie systému. Zmenou premenných v simulácii je možné sledovať správanie systému. Modelovanie systémov sa tradične uskutočňuje prostredníctvom matematického modelu, ktorý sa pokúša nájsť analytické riešenia umožňujúce predikovať správanie systému zo súboru parametrov a počiatočných podmienok. Počítačová simulácia sa stala

užitočnou súčasťou modelovania mnohých fyzikálnych, chemických, biologických, priemyselných, ekonomických, a iných druhov systémov, aby človek získal prehľad a poznatky o ich fungovaní za rôznych inicializačných podmienok. Existuje mnoho rôznych typov počítačových simulácií. Spoločnou vlastnosťou všetkých typov je však snaha vytvoriť reprezentatívne vzorky scenárov pre model, ktoré by vymenovali všetky možné, poprípade aj nemožné stavy systému.

Robotický simulátor je jedným z typov počítačových simulátorov slúžiaci na vývoj aplikácií pre virtuálne modely robotov bez potreby vytvárania ich fyzických modelov. Spoločnou vlastnosťou robotických simulátorov s vizualizáciou je 2D alebo 3D modelovanie, čo súvisí s vykresľovaním robota, ale aj jeho prostredia. Popri vývoji aplikácií pre mobilné roboty a statické roboty s pevne viazanou základňou, umožňujú používateľovi vytvárať jednoduché, ale aj komplexné svety pozostávajúce z jedného alebo viacerých virtuálnych modelov robotov, svetelných zdrojov a hmotných predmetov prostredia. Významnou vlastnosťou, ktorá odlišuje vizualizáciu od simulácie je prítomnosť fyzikálneho enginu. Fyzikálny engine dodáva simulovanému prostrediu pôsobenie síl – tiažová sila, sila akcie a reakcie, vzájomné kolízie objektov a pod. čím je dosiahnuté viac realistickejšie prevedenie[4].

Robotický simulátor umožňuje písanie programov pre robota v off-line móde. Úspech offline programovania však závisí od toho, nakoľko je reálne prostredie robota podobné prostrediu simulácie. Existujú prípady, kedy môžu byť vyvíjané aplikácie z virtuálneho modelu simulácie prenesené priamo na fyzický model bez väčších úprav zdrojového kódu. Akcie robota založené na senzoroch sa simulujú v off-line režime oveľa ťažšie, pretože pohyb robota závisí od okamžitých hodnôt senzorov v reálnom svete[5].

Rozdiely medzi vizualizáciou a simuláciou v robotike:

- Simulácia umožňuje definovať rôzne počiatočné stavy systému pre vyvíjaný model robota.

- V simulácii sa dokáže mobilný robot pohybovať v priestore a interagovať s

prostredím.

- Simulácia nemusí podporovať vkladanie abstraktných objektov napr. trajektórie.

- Vizualizácia sa sústreďuje len na zobrazenie stavu samotného modelu.

- Vo vizualizácii je absencia prítomnosti fyzikálneho enginu – vo vizualizovanom prostredí neexistuje pôsobenie gravitácie a vzájomná kolízia objektov a robota.

- Vizualizačný softvér je menej náročný na výpočtový výkon zariadenia v porovnaní s komplexnými simulačnými nástrojmi s fyzikálnym enginom.

Výhody používania vizualizácie a simulácie v robotike:

- Možnosť zobrazenia abstraktných objektov, ktoré v reálnom svete nie je možné reprezentovať.

- Nie je potrebné zakúpiť hardvér fyzického robota, ktorý je častokrát finančne nákladný.

- Nie sú potrebné znalosti pre prácu s hardvérom fyzického robota.

- Rýchlejšie testovanie navrhnutého dizajnu robota.

- Rýchlejší návrh riadiaceho softvéru.

- Rýchlejšie testovanie funkcionality riadiaceho softvéru (uplatňované hlavne pri využívaní prvkov umelej inteligencie).

- Nie je potrebné navrhovať ovládače fyzického hardvéru.

- Nie je potrebný častý servis opotrebovaných komponentov pri nadmernej práci s robotom.

- Možnosť rýchleho kopírovania rovnakého typu robota (využívané hlavne pri rojovej – *swarm* robotike alebo multiagentovej robotike).

- Možnosť simulovania prostredia pre robota, ktorého dosiahnutie by bolo v reálnom prostredí finančne nákladné.

- Bezpečnosť – pri práci s niektorými typmi robotov môže dôjsť k poškodeniu robota alebo v určitých prípadoch aj ublíženiu na zdraví človeka.

Nevýhody používania vizualizácie a simulátora v robotike:

- Človek stráca možnosť získať nové skúsenosti pri práci s fyzickým robotom a jeho hardvérom.

- Bezproblémové fungovanie softvéru robota a virtuálneho modelu robota nezabezpečuje to, že aj jeho fyzický model bude fungovať rovnako.

- Nemožnosť reprezentovať reálny stav opotrebovania komponentov robota napríklad opotrebovanie ozubenia motorov, opotrebovanie kapacity batérie a pod.

- Absencia vytvorenia reálnych podmienok dynamicky sa meniaceho fyzického prostredia, ktoré sú častokrát nepredvídateľné.

2.2 Grafické užívateľské rozhranie

Grafické užívateľské rozhranie (GUI) je forma užívateľského softvérového rozhrania (UI), ktorá umožňuje užívateľom interagovať s elektronickými zariadeniami prostredníctvom grafických prvkov ako sú okná, tlačidlá, posuvníky, rozbaľovače, textové polia a pod.

Ich používanie je pre užívateľa intuitívnejšie a jednoduchšie na naučenie narozdiel od textových používateľských rozhraní – príkazový riadok (CLI), ktoré vyžadujú manuálne zadávanie príkazov na klávesnici počítača. Pri zadávaní príkazu alebo sekvencie príkazov do CLI môžu vzniknúť syntaktické chyby, ktoré musí programátor ošetriť. Pre užívateľa to zase znamená, že musí opätovne zadať požadovaný príkaz alebo ich sekvenciu, zatiaľ čo v prípade GUI túto pracnú činnosť možno nahradiť

napríklad kliknutím tlačidla na obrazovke. Akčný zásah do GUI je zvyčajne vykonávaný priamou manipuláciou s grafickými prvkami za použitia vstupných periférií napríklad myš, dotyková obrazovka, trackball, klávesnica a pod[6].

Okrem grafických operačných systémov pre stolné počítače a laptopy sa GUI implementujú v mobilných zariadeniach, ako sú smartfóny, prenosné prehrávače médií, prehrávače MP3, herné zariadenia alebo menšie domáce, kancelárske a priemyselné elektronické zariadenia a mnoho ďalších.

Pred začatím programovania GUI aplikácie je potrebné zvoliť si vhodný programovací jazyk alebo platformu, ktorá najlepšie spĺňa požiadavky na implementáciu. Požiadavkami kladenými na vývoj vybraného typu aplikácie sú platforma, pre ktorú bude aplikácia určená (mobilná, desktopová alebo webová aplikácia) a existencia knižníc podporujúcich rýchlu tvorbu GUI prostredníctvom predprogramovaných grafických prvkov.

V oblasti robotiky sa GUI využíva ako riadiaci a vizualizačný interface medzi človekom a strojom – robotom (z angl. HMI *Human Machine Interface*). Rozhraním tak dokáže ovládať robota aj osoba bez programátorských skúseností. Grafickými prvkami GUI je možné vykonávať diaľkovo riadené pohyby robota, akčné zásahy do chodu programu, nastavovať a vizualizovať vnútorné parametre. Nevýhodou vytvárania aplikácie grafického rozhrania pre konkrétny typ robota je nemožnosť jeho použitia na iný.

2.3 Softvérový balík

Rozsiahle programové projekty si vyžadujú vytvorenie usporiadanej štruktúry adresárov a súborov. Softvérový balík je skupina na seba nadväzujúcich súborov a programov, ktoré majú spoločný logický súvis. Pokiaľ je balík určený pre vývoj alebo individuálne prispôsobenie, môže navyše obsahovať zdrojové kódy, vzorové súbory, spustiteľné aplikácie, dokumentáciu k zdrojovým kódom alebo fungovaniu súčastí balíka a manuály k ovládaniu aplikácií. Všetky komponenty softvérového

balíka môžu byť určené na výrazne odlišné veci, ale pri spúšťaní alebo kompilovaní aplikácie sú komponenty spojené do jedného funkčného celku[7].

Balík nemusí obsahovať úplne všetky zdroje potrebné na spustenie jeho obsahu, pokiaľ existuje spôsob, ako nezahrnuté zdroje externe dodať používateľovi, poprípade sú súčasťou operačného systému (externé knižnice, externé programy, kompilátory, interpretery atď.). Tým je zabezpečená kompaktnosť – nižšia náročnosť na úložný priestor a rýchlejší transport balíka zo zariadenia na zariadenie [8]. Na druhú stranu programátor by mal pri publikácii balíka dodať potrebnú dokumentáciu, ktorá obsahuje zoznam všetkých použitých externých zdrojov a ich verzií pri spúšťaní jeho obsahu. Programátor pri vytváraní softvérového balíka môže využívať dva prístupy:

1. Jeden hlavný program, ktorý zahŕňa menšie podprogramy.

   Balíky, ktoré zapuzdrujú všetky svoje funkcie do jednej veľkej aplikácie, sú užitočné pre väčšinu užívateľov. Výhodami tohto prístupu sú ľahká inštalácia, pochopiteľnosť a spustenie. Aplikácia nemusí mať fixnú funkcionalitu a môže užívateľovi dovoľovať stiahnuť doplnky podľa jeho vlastných preferencií a potrieb. Celý proces inštalácie doplnkov sa deje na pozadí s minimálnou interakciou a úsilím užívateľa.

2. Softvérový balík s roztrieštenou funkcionalitou obsahu.

   Tento typ balíka je zameraný pre programátorsky zdatnejších používateľov. Fragmentované balíky môžu obsahovať nástroje príkazového riadku spolu s rozsiahlou dokumentáciou a flexibilnými možnosťami prispôsobovania. Zmeny v týchto balíkoch sa zvyčajne musia vykonať buď manipuláciou s adresárovými štruktúrami, alebo úplným stiahnutím novej verzie balíka.

Vlastnosťami zdrojových kódov softvérových balíkov sú:

1. Modulárnosť

   Kód je navrhnutý technikou modulárneho programovania, ktorá rozdeľuje funkcionalitu na menšie nezávislé moduly – *top-down* prístup a technikou objek-

tovo orientovaného programovania, čo znamená, že každá dátová štruktúra obsiahnutá v module je zapísaná ako trieda. Spájaním modulov a vytváraním rôznych kombinácií je možné dosiahnuť požadované správanie výslednej aplikácie.

2. Rozšíriteľnosť o funkcionalitu

   Modul predstavuje logický element a jeho dátové štruktúry sú schopné komunikovať s dátovými štruktúrami iných modulov, pokiaľ je medzi nimi dátový súvis. Dodáva špecifickú časť funkcionality vytváranej aplikácie, na ktorú bol určený. Ak je potrebné rozšíriť softvérový balík o konkrétnu funkcionalitu, stačí pridať nový modul s príslušným zdrojovým kódom.

3. Prehľadnosť

   Každý modul v balíku má svoj vlastný súbor, čím sa oddelí jeho kód od kódu ostatných modulov. Objektovo orientované programovanie enkapsuluje premenné a metódy, ktoré navzájom spolu súvisia do tried vrámci príslušného modulu a zabezpečí sa ich vzájomné previazanie.

4. Univerzálnosť

   Logika modulov je programovaná tak, aby ich bolo možné implementovať aj pri riešení podobných typov problémov a to bez väčších úprav zdrojových kódov.

5. Selektívnosť

   Možnosť použitia len vybraných modulov podľa požiadaviek na vyvíjanú aplikáciu.

Ak je balík vytváraný na konkrétny typ problému musí obsahovať konfiguračné súbory prispôsobené danej aplikácii. Vlastnosťami konfiguračných súborov sú:

1. Štandardizácia

Súbory sú napísané v určitom štandardnom formáte napríklad JavaScript Object Notation (JSON), eXtensible Markup Language (XML), Comma Separated Value (CSV) a pod. Výhodou serializácie súborov inicializačných hodnôt v štandardných formátoch je možnosť použitia existujúcich knižníc so syntaktickým analyzátorom prispôsobeným na deserializáciu zvoleného formátu do dátových štruktúr.

2. Čitateľnosť

   Textová podoba obsahu súborov zabezpečuje ľahkú čitateľnosť pre toho, kto by chcel vybraný súbor editovať podľa vlastných potrieb.

3. Rekonfigurácia po kompilácii

   Zdrojový kód vyvíjanej aplikácie je oddelený od vstupných inicializačných konštánt a tým sa zabezpečí rýchla rekonfigurácia. V prípade kompilovaných aplikácií už nie je možné meniť hodnoty, ak boli nastavené v jej zdrojových kódoch. Rekonfigurácia by v takom prípade bola možná, iba ak by sa aplikácia opätovne rekompilovala s novonastavenými hodnotami v súboroch zdrojových kódov za predpokladu, že sú zdrojové kódy voľne dostupné[7].

Usporiadanie súborov do softvérového balíka je možné využiť aj v robotickom softvéri. Tým sa vytvorí vhodné prostredie na vývoj aplikácií pre virtuálny alebo fyzický model robota.

3 Prehľad implementovaných technológií

Táto kapitola je zameraná na detailný popis konkrétnych technológií, ktoré sú implementované v realizácii softvéru.

3.1 Open3D

Robotika a počítačová 3D grafika potrebovala pre vývoj knižnicu, ktorá by bola schopná rýchlo spracovať dáta formátov 3D súborov napríklad súbory **.stl*, **.obj*, **.pcd* a manipulovať s nimi za použitia grafického procesora (GPU) v počítači.

Knižnica Open3D bola vytvorená na riešenie tohto problému. Vývojári na nej pracujú od roku 2015 a bola použitá v množstve publikovaných výskumných projektov. Podľa [9], dovtedy neexistovala žiadna tzv. *open source* knižnica, ktorá by bola rýchla, jednoduchá na použitie, schopná efektívne načítať a vizualizovať 3D dáta pomocou niekoľkých riadkov kódu v súlade s modernými postupmi softvérového inžinierstva. Open3D podporuje rýchly vývoj softvéru pracujúceho s 3D dátami. Vývojári optimalizovali backend knižnice využitím paralelizácie aplikačného programového rozhrania (API) OpenMP a implementovania zdrojových kódov v štandarde C++ 11. Knižnica podporuje načítavanie dát zo súborov do dátových štruktúr troch štandardných reprezentácií 3D objektov:

1. *point cloud* – mračná bodov,
2. *triangle meshes* – trojuholníkové sieťoviny,
3. RGB-D *images* – RGB obraz s dodatočnou súradnicou hĺbky.

Open3D má implementovanú kompletnú sadu základných algoritmov spracovania 3D dát, ako sú algoritmy čítania a zápisu, algoritmy zmeny vzorkovania, algoritmy konverzie medzi formátmi a funkciu vizualizácie spustiteľnej v okne operačného systému. Kľúčovou vlastnosťou pre zvolenie tejto knižnice je implementácia v jazyku Python. Backend je naprogramovaný v jazyku C++, zatiaľ čo dátové štruktúry a

funkcie v jazyku Python predstavujú frontendové rozhranie. Vývojárovi aplikácie postačuje použiť jazyk Python na načítanie 3D objektov do vizualizácie a manipulovať tak s vloženými 3D objektami prostredníctvom API. Okrem programovania kódu v jazyku Python, knižnica umožňuje programátorom využiť vizualizáciu v Jupyter notebooku[9].

3.2 Webots

Webots je open source a multiplatformová desktopová aplikácia určená na simuláciu robotov vo virtuálnom prostredí. Podľa [10] aplikácia poskytuje kompletné vývojové prostredie na modelovanie, programovanie a simuláciu robotov. Simulátor je navrhnutý pre profesionálne použitie v modelovaní a simuláciách v priemysle, výskume a vzdelávaní. Švajčiarska spoločnosť Cyberbotics Ltd. spravuje a pravidelne každý polrok aktualizuje simulátor Webots ako svoj hlavný produkt od roku 1998. Aplikácia je založená na kombinácii moderného grafického užívateľského rozhrania Qt, fyzikálneho enginu Open Dynamics Engine (ODE) a vykresľovacieho enginu OpenGL 3.3.

Aplikáciu je možné nainštalovať iba na 64-bitových operačných systémoch:

- Windows (7, 8, 8.1, 10),
- Linux (Ubuntu 12.04 a vyššie, Debian, RedHat, Mandrake, Gentoo, SuSE),
- macOS (10.12, 10.13).

Prostredie simulátora poskytuje prostriedky na vytvorenie sveta a modelov robotov podľa vlastných potrieb. Okrem týchto prostriedkov vlastnej tvorby dáva možnosť vyskúšať existujúce simulácie modelov robotov, celé svety s modelmi robotov a ich predprogramovanými vzorovými ovládačmi [10].

Realizácia modelovania virtuálneho robota je možná z prvkov, ktoré sú v ponuke prostredia:

- robot

- virtuálne vstupné periférie – senzory
- virtuálne výstupné periférie – akčné členy
- pevné telesá
- tvary
- transformácie
- spoje
- fyzika

Sieťoviny tvarov objektov a kolíznych objektov je možné vkladať prostredníctvom preddefinovaných 3D modelov primitív (kváder, valec, guľa, kužeľ, kapsula a pod.) alebo importovaním komplexných 3D modelov vytvorených v CAD softvéri napríklad AutoCAD, Tinkercad, Fusion 360, a pod. exportovaných do súboru stereolitografického formátu (STL)[11].

Zdrojový kód ovládača virtuálneho robota môže byť napísaný v jazykoch C, C++, Python, Java, MATLAB. Jazyky C++, Python a Java na prístup k dátovým štruktúram simulácie využívajú prvky objektovo-orientovaného programovania. Každý virtuálny senzor, akčný člen alebo iný prvok simulácie má vytvorenú triedu s príslušnou funkcionalitou. V jazykoch C, a MATLAB sa na prístup k dátovým štruktúram simulácie využívajú prvky procedurálneho programovania. Každý virtuálny senzor, akčný člen alebo iný prvok simulácie má vytvorené statické funkcie s príslušnou funkcionalitou[12].

3.3 WPF

Windows Presentation Foundation (WPF) je softvérový rámec užívateľského rozhrania (z angl. User Interface – UI) určený na tvorbu GUI v podobe klientskych aplikácií pre stolné počítače a laptopy s operačným systémom Windows. Vývojárska platforma WPF podporuje širokú škálu funkcií pre tvorbu aplikácií vrátane

aplikačných modelov, zdrojov, ovládacích prvkov a ich rozloženia, grafiky, dátového previazania, dokumentov a zabezpečenia. Okrem existujúcich grafických prvkov je možné dedením vlastností základných prvkov vytvárať vlastné grafické prvky s požadovanou funkcionalitou. WPF je súčasťou softvérového rámca .NET Framework. Pokiaľ programátor disponuje znalosťami vývoja aplikácií v ASP.NET alebo Windows Forms, dokáže ich uplatniť aj pri vývoji aplikácií vo WPF[13].

Grafika WPF, vrátane prvkov plochy okna a samotných okien, je vykresľovaná prostredníctvom API Direct3D (D3D). Rozhranie D3D umožňuje zobrazovať komplexnejšiu grafiku a umožňuje operačnému systému Windows presunúť výpočet grafického vyobrazenia na GPU. Tým sa znižuje zaťaženie na procesor základnej jednotky (CPU) počítača.

3.3.1 XAML

WPF je navyše rozšírený o značkovací jazyk XAML (eXtensible Application Markup Language). XAML je postavený na značkovacom štandarde jazyka XML podobne ako značkovací jazyk HTML (HyperText Markup Language), ktorý je určený na štrukturalizáciu webových stránok[14]. Použitie XAML na vývoj používateľských rozhraní oddeľuje grafickú stránku aplikácie od aplikačnej logiky - frontend od backendu. Táto vlastnosť je považovaná za štandardný prístup pri návrhu architektúry grafického rozhrania. XAML poskytuje deklaratívny model pre tvorbu dizajnu aplikácie rozhrania. To umožňuje vývojárovi alebo dizajnérovi opísať správanie a integráciu jednotlivých komponentov bez použitia procedurálneho programovania. Pri kompilácii sa zdrojový kód všetkých elementov a ich atribútov napísaných v XAML mapuje na príslušné inštancie tried a ich vlastnosti do jazyka C#[15].

3.3.2 Animácie

WPF podporuje tvorbu animácií postavených na časovom prístupe narozdiel od prístupu založenom na snímkach. Táto vlastnosť oddeľuje rýchlosť chodu animácie od výkonu zariadenia a animácia bude trvať takmer rovnako dlho na menej výkon-

nom zariadení ako aj na výkonnejšom zariadení. Ďalej podporuje animácie nízkej úrovne prostredníctvom časovačov a abstrakciu animácií vyššej úrovne prostredníctvom tried *Animation*. Animovať je možné ľubovoľnú vlastnosť grafického prvku WPF pokiaľ je definovaná ako *DependencyProperty*. Animácie je možné zoskupiť do inštancií tried *Storyboard*. Tieto inštancie sú primárny spôsob ako spúšťať, zastavovať, pozastavovať alebo ináč manipulovať s chodom animácií. Animácie môžu byť spustené rôznymi externými udalosťami vrátane akcie používateľa, ale aj internými udalosťami vrámci aplikácie ako sú napríklad časovače. Časovače animácií sú inicializované a spravované WPF softvérovým rámcom. Inštancie animácií možno definovať ako XAML elementy alebo prostredníctvom jazyka C#[15].

Používanie animácii pri tvorbe aplikácií s grafickým rozhraním dizajnovo vylepší grafickú stránku aplikácie, zmierni rýchlosti prechodov z jedného stavu do druhého (napr. prepínanie medzi oknami) alebo zdôrazní významnosť nejakého elementu (napr. vstup kurzoru myši do oblasti tlačidla), no neprináša žiadnu novú funkcionalitu do existujúcej aplikácie, iba zvyšuje nároky výpočtového výkonu zariadenia, na ktorom bude aplikácia spustená.

3.3.3 Material Design Themes

Material Design Themes je knižnica rozširujúca dizajn softvérového rámca WPF. Inštalácia knižnice je možná prostredníctvom .NET správcu balíkov NuGet.

Vlastnosti knižnice:

- Transformácia dizajnu existujúcich WPF grafických ovládacích prvkov zo štandardného nastavenia dizajnu na viac zákaznícky orientovaný dizajn.

- Rozšírenie štandardného množstva existujúcich ovládacích prvkov o ďalšie dodatočné prvky s príslušnou funkcionalitou.

- Možnosť jednoduchého nastavenia farebných paliet Material Design.

3.4 Socket

Sieťový socket je dátová štruktúra sieťového uzla – koncového zariadenia počítačovej siete napr. desktopového počítača, laptopu, servera, mobilného zariadenia a pod. Koncovému zariadeniu slúži ako softvérový bod odosielania a prijímania dát v sieti. Štruktúru a vlastnosti socketu definuje API sieťovej architektúry. Sockety sa vytvárajú, iba ak je proces aplikácie spustený na zariadení. Kvôli štandardizácii protokolov Transmission Control Protocol/Internet Protocol (TCP/IP) pri vývoji internetu sa pojem sieťový socket najčastejšie používa v kontexte sady internetových protokolov. Preto sa často označuje aj ako internetový socket. Termínom socket sú označované aj body komunikácie interných procesov (IPC) vrámci jedného zariadenia, ktoré často používajú rovnaké API ako sieťové sockety[16]. V týchto ponímaniach je socket externe alebo interne identifikovaný pre ostatných hostiteľov trojicou parametrov:

1. transportnný protokol,
2. IP adresa (IPv4 alebo IPv6),
3. číslo portu (0-65535).

Sockety sa podľa prednastaveného transportného protokolu rozdeľujú na:

1. datagramové sockety,
2. streamové sockety.

Výhodou používania socketov pri komunikácii dvoch alebo viacerých aplikácií sú štandardy definované protokolmi. Komunikačné rozhranie je tak nezávislé od programovacieho jazyka, v ktorom sú zdrojové kódy komunikujúcich aplikácií napísané. Táto vlastnosť má podstatnú úlohu pri vývoji softvéru, pretože programátorovi dovoľuje zvoliť si vhodný programovací jazyk podľa požiadaviek na aplikačný softvér[16].

3.4.1 Datagramové sockety

Datagramové sockety využívajú protokol transportnej vrstvy User Datagram Protocol (UDP). Každý paket odoslaný alebo prijatý datagramovým socketom je adresovaný a smerovaný individuálne. V komunikácii využívajúcej UDP vystupujú dva procesy – odosielací proces a prijímací proces. Pokiaľ je medzi odosielacím a prijímacím procesom poloduplexný tok dát, potom komunikácia prebieha v piatich bodoch:

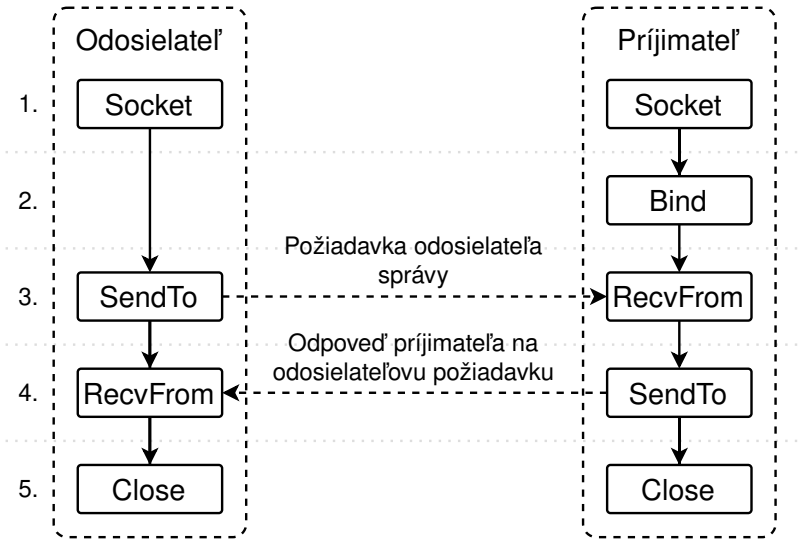
1. Vytvorenie dátovej štruktúry socketu na oboch stranách.
2. Previazanie socketu príjimacieho procesu s IP adresou a portom.
3. Odoslanie UDP paketu odosielacím procesom s požiadavkou na adresu a port prijímacieho procesu.
4. Prijatie UDP paketu odosielacím procesom s odpoveďou z adresy a portu prijímacieho procesu.
5. Uzatvorenie socketov.

Sekvenčný diagram UDP komunikácie medzi prijímacím a odosielacím procesom s využitím datagramových socketov zobrazuje obrázok (3 – 1).

Transportovaný UDP paket má 8-bajtovú hlavičku, ktorá sa skladá zo štyroch 16-bitových hodnôt. Obsahom štruktúry hlavičky sú:

1. zdrojový port – identifikuje odosielací proces spustený na zariadení,
2. port destinácie – identifikuje prijímací proces spustený na zariadení,
3. dĺžka – veľkosť UDP paketu v bajtoch,
4. kontrolný súčet – pokrýva hlavičku aj prenášané dáta

Štruktúru hlavičky UDP paketu znázorňuje obrázok (3 – 2).



Obrázok 3 – 1 Sekvenčný diagram UDP komunikácie

Hlavička UDP paketu																														
Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0-31	Zdrojový port																Port destinácie															
32-63	Dĺžka																Kontrolný checksum														
64-...	Dáta																															

Obrázok 3 – 2 Štruktúra hlavičky UDP paketu

Použitie datagramového socketu nezaručuje spoľahlivosť prijatia odoslaného paketu. V praxi to znamená, že prijímací proces nemusí úspešne dostať transportovaný paket [17]. Viacero paketov odoslaných z jedného zariadenia na druhé alebo z procesu do procesu vrámci jedného zariadenia môže prísť v ľubovoľnom poradí alebo nemusia dôjsť vôbec. Na druhú stranu, výpočtová náročnosti spracovania UDP paketu je oveľa nižšia pre koncové zariadenie narozdiel od Transmission Control Protocol (TCP) paketu a tým pádom je dosiahnutá vyššia prenosová rýchlosť. Datagramové sockety sa primárne využívajú v aplikáciách streamujúcich dáta v reálnom čase – video a zvuk, pri ktorých prevádzkovaní je vyžadovaná nízka latencia a spoľahlivosť prijatia paketov nehrá dôležitú úlohu.

3.4.2 Streamové sockety

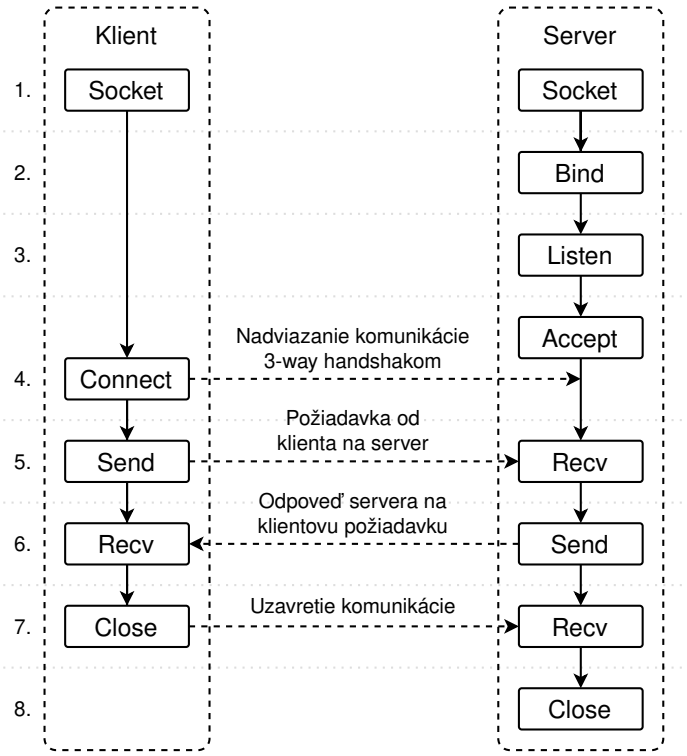
Streamové sockety sú orientované na pripojenia využívajúce protokoly transportnej vrstvy:

- TCP (Transmission Control Protocol)
- SCTP (Stream Control Transmission Protocol)
- DCCP (Datagram Congestion Control Protocol)

Sockety si vzájomne preposielajú sekvenované toky dát bezstratovo, s jasne definovanými mechanizmami na otváranie a uzavieranie prepojení a hlásenie chýb v pripojení. V komunikácii využívajúcej protokol TCP vystupujú dva entity – server a klient[17]. Pokiaľ je medzi serverom a klientom poloduplexný tok dát potom komunikácia prebieha v ôsmich bodoch:

1. Vytvorenie dátovej štruktúry socketu na oboch stranách.
2. Previazanie serverového socketu s IP adresou a portom.
3. Čakanie na prichádzajúceho klienta.
4. Akceptovanie klientského socketu pomocou trojcestného handshakingu z angl. *three-way handshake*.
5. Odoslanie TCP paketu klientom s požiadavkou na server.
6. Prijatie TCP paketu na strane klienta s odpoveďou zo servera.
7. Uzatvorenie klientského socketu a notifikácia serverového socketu o tejto akcii.
8. Uzatvorenie serverového socketu.

Sekvenčný diagram TCP komunikácie medzi serverom a klientom s využitím streamových socketov zobrazuje obrázok (3 – 3).



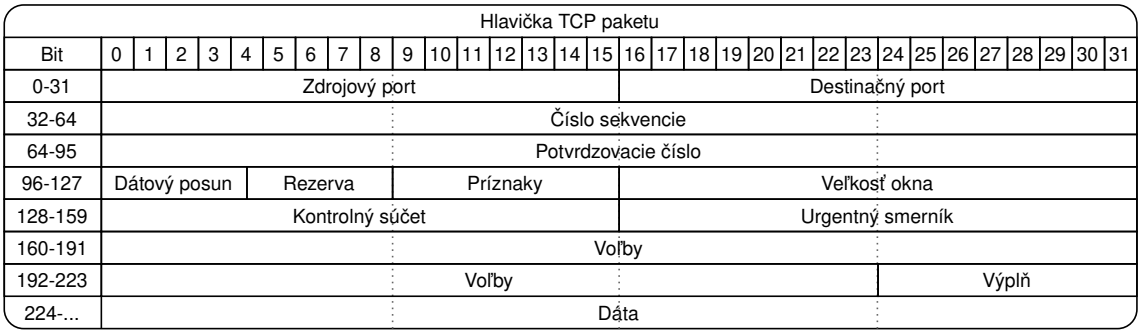
Obrázok 3 – 3 Sekvenčný diagram TCP komunikácie

Hlavička transportovaného TCP paketu sa skladá z dvanástich polí, ale jej veľkosť sa môže líšiť v závislosti od nastavenia komunikácie. Minimálna veľkosť je 20 bajtov a maximálna 60 bajtov. Obsahom štruktúry paketu sú:

1. zdrojový port – 16-bitový, identifikuje odosielací proces spustený na zariadení,
2. port destinácie – 16-bitový, identifikuje prijímací proces spustený na zariadení,
3. číslo sekvencie – 32-bitové,
4. potvrdzovacie číslo – 32-bitové,
5. dátový posun – 4-bitové, veľkosť hlavičky po 32 bitoch označovaných ako slová,
6. rezerva – 3-bitová, aktuálne nevyužívané bity nastavené na hodnotu nula,
7. príznaky – 9 kontrolných bitov,

8. veľkosť okna – 16-bitová,
9. kontrolný súčet – 16-bitový, slúži na detekciu chýb vrámci hlavičky TCP, IP pseudohlavičky a samotných prenášaných dát,
10. urgentný smerník – 16-bitový,
11. voľby – veľkosť je variabilná od 0 po 40 bajtov v závislosti od nastavenia komunikácie,
12. výplň – veľkosť je variabilná, doplňa pole volieb do 32 bitov nulovými bitmi.

Štruktúru hlavičky TCP paketu znázorňuje obrázok (3 – 2).



Obrázok 3 – 4 Štruktúra hlavičky TCP paketu

Použitie streamových socketov zaručuje spoľahlivosť prijatia odoslaného paketu. Pakety sú prenášané bezstratovo narozdiel od datagramových socketov[18]. Opakovaným odosielaním stratených alebo poškodených paketov cez nespoľahlivú internetovú sieť preventívne zabezpečuje pred náhodným usporiadaním prijatých paketov. Tým sa zaisťuje ich prijatie v správnom poradí. Veľkosť hlavičky paketu a jeho bezpečný transport ovplyvňuje rýchlosť prenášaných dát, ktorá je podstatne nižšia a latencia zasa vyššia narozdiel od transportu UDP paketu. V dnešnej dobe rýchleho internetu, kedy sa prenosové rýchlosti pohybujú od $100Mb/s$ až do $1Gb/s$ je latencia TCP v porovnaní s UDP zanedbateľná. Internet, fungujúci na štandarde TCP/IP, implementuje práve tieto streamové TCP sockety vo všetkých oblastiach od sťahova-

nia súborov, cez prenos hypertextových súborov webových stránok až po odosielanie dát medzi IoT (Internet of Things) zariadeniami a cloudami v sieti [19].

4 Knižnica Core

Pre prácu so softvérovými balíkmi kĺbových robotov bola vytvorená knižnica *Core* so zdrojovými kódmi napísanými v jazyku Python. Obsahuje základné moduly s dátovými štruktúrami, ktoré majú spoločné využitie pri vytváraní aplikácií pre všetky typy kĺbových robotov:

1. *Communication.py*,
2. *Kinematics.py*,
3. *Regulators.py*,
4. *Timers.py*,
5. *Visualization.py*,
6. *Package.py*.

4.1 Modul Communication.py

Prvý modul *Communication.py* je zameraný na realizáciu komunikácie medzi dvoma spustenými aplikáciami prostredníctvom rozhrania socket využívajúceho TCP protokol transportnej vrstvy. Streamové sockety boli zvolené kvôli ich spoľahlivému transportu paketov, schopnosti vzájomnej notifikácie aplikácií pri ukončení prepojenia a architektúre klient-server. Všetky dáta prenášané medzi klientskými a serverovými aplikáciami sú v štandardnom formáte JSON. Výhodami používania formátu JSON na prenášané dáta sú:

- jednoduchý syntax,
- čitateľnosť,
- kompaktnosť,
- rýchly prenos,

- možnosť využívať vnáranie,

- možnosť využiť existujúce knižnice na serializáciu a deserializáciu dát.

Sockety sú komponentom dvoch tried vystupujúcich v komunikácii:

1. *JsonServer* – server,

2. *JsonClient* – klient.

JsonServer je trieda predstavujúca komunikačný bod serverového programu s pripojeným klientom prostredníctvom TCP paketov. Jedna inštancia triedy *JsonServer* dokáže synchrónne obsluhovať iba jedného pripojeného klienta. Akceptovaný socket je po úspešnom pripojení klienta nastavený na neblokovaný režim. Výhodou neblokovaného režimu je možnosť programu pokračovať v hlavnej slučke bez pozastavenia hlavného vlákna procesu a čakania na prijatie paketu od klienta do čítacej vyrovnávacej pamäte. Obsah odosielaných paketov je formatovaný v štandarde JSON. JSON objekt serializovaný klientom do obsahu paketu má pevne definovanú formu, ktorá musí byť zachovaná, aby dokázal *JsonServer* na základe neho vyhľadať príslušný smerník funkcie v asociatívnom poli. Prijatý paket je dekódovaný JSON dekóderom a musí obsahovať nasledujúce kľúče:

1. *Topic*

   - téma alebo kľúčové slovo prvého stupňa vnárania v asociatívnom poli, pod ktorým má server hľadať hodnotu druhého stupňa vnárania,

   - hodnota kľúča je definovaná programátorom podľa názvu funkcie, ktorá je nastavená ako koncový bod,

2. *Method*

   - typ metódy alebo kľúčové slovo druhého stupňa vnárania v asociatívnom poli, pod ktorým má server hľadať smerník na funkciu,

   - nadobúda iba pevne definované hodnoty:

(a) GET – čítanie,

(b) SET – zápis,

(c) UPDATE – aktualizácia,

3. *Content*

- obsah, ktorý je vstupným parametrom do koncovej funkcie pri jej volaní,
- serializovaný JSON objekt.

JsonServer obsahuje internú premennú *root* dátového typu asociatívneho pola. Premenná *root* má vždy dvojstupňové vnáranie, aby bola zabezpečená kompatibilita s prijatým paketom. Funkcia je zavolaná po prijatí paketu na klientskom sockete, ktorý sa na ňu dopytuje prostredníctvom kľúčov témy a metódy. Nasledujúci text zobrazuje príklad konzolového výstupu pri vypísaní obsahu štruktúry premennej *root* s jedným definovaným koncovým bodom.

```
>>> { # JsonServer.root
        "Origin": {
            "GET": <function GetOrigin at 0x0000020B77618CA0>
        }
    }
```

Prvý reťazec *"Origin"* je kľúč prvej úrovne. Druhý reťazec *"GET"* je kľúč druhej úrovne. Poslendý reťazec *"<function GetOrigin at 0x0000020B77618CA0>"* je hexadecimálny zápis adresy smerníka funkcie s názvom *GetOrigin* v pamäti počítača. Pridanie nového smerníka na funkciu do asociatívneho poľa *root* je možné prostredníctvom metódy *AddEndPoint*, ktorá využíva návrhový vzor dekorátor. Dekorovaná funkcia touto metódou musí mať práve jeden vstupný parameter. Návratová hodnota funkcie môže, ale nemusí byť definovaná. Preddefinovaná návratová hodnota funkcií v jazyku Python je *None* objekt, čo predstavuje prázdnu odpoveď. Podľa štandardu JSON sa Python objekt *None* serializuje na JSON objekt *null*. Nasledujúci príklad

zdrojového kódu zobrazuje pridanie smerníka funkcie *SetOrigin* s návratovou hodnotou dátového typu *list* do asociatívneho poľa s kľúčom prvého stupňa *Origin* a kľúčom druhého stupňa *SET* dekoračnou metódou *AddEndPoint*.

```
@JsonServer.AddEndPoint(topic="Origin", method="SET")
def SetOrigin(content):
    return [[0.0]*3]*6
```

Nasledujúci text zobrazuje vypísanú štruktúru asociatívneho poľa *root* do konzolového výstupu s novo-pridaným smerníkom funkcie *SetOrigin* po aplikovaní dekoračnej metódy.

```
>>> { # JsonServer.root
        "Origin": {
            "GET": <function GetOrigin at 0x0000020B77618CA0>
            "SET": <function SetOrigin at 0x0000020B77618CB8>
        }
    }
```

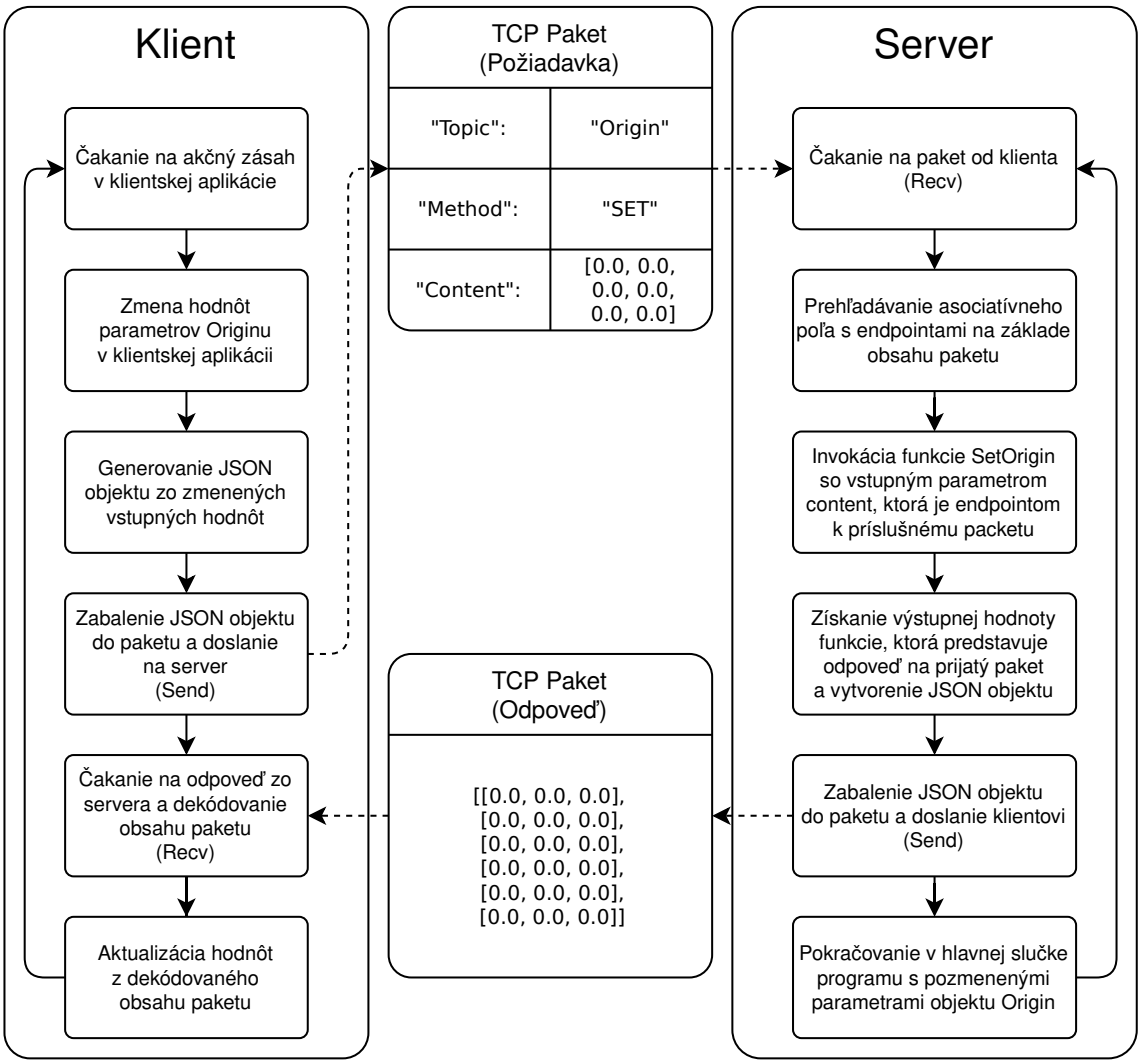
Nasledujúci text zobrazuje príklad obsahu paketu serializovaného objektu do formátu JSON, ktorý prijal server od klienta ako požiadavku.

```
{ # Poziadavka
    "Topic": "Origin",
    "Method": "SET",
    "Content": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
}
```

Server po dekódovaní obsahu prijatého paketu zavolá metódu, ktorá sa nachádza v asociatívnom poli koncových bodov na kľúči prvého stupňa *Origin* a následne na kľúči druhého stupňa *SET* so vstupným parametrom poľa šiestich hodnôt dátového typu *float*. Odpoveďou servera na prijatý paket je výstupná hodnota invokovanej funkcie *SetOrigin* serializovaná do formátu JSON. Nasledujúci text zobrazuje odpoveď obsahu serializovaného paketu, ktorý odoslal server klientovi ako odpoveď.

```
# Odpoved
[[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]
```

Obrázok (4 – 1) zobrazuje sekvenčný diagram vyššie popisovanej príkladovej komunikácie medzi klientom a serverom.



Obrázok 4 – 1 Sekvenčný diagram komunikácie

4.2 Modul Kinematics.py

Druhým modulom je *Kinematics.py*, ktorý je optimalizovanou verziou modulu kinematiky z mojej bakalárskej práce. Štruktúra kinematického modelu robota je rovnako ako v module bakalárskej práce načítaná prostredníctvom súboru formátu XML, avšak s menšími zmenami názvov značiek a ich atribútov. Optimalizácia kódu bola nevyhnutná z dôvodu pomalého výpočtu inverznej kinematiky. Pomalý výpočet koncových bodov končatín by znižoval obnovovaciu frekvenciu vizualizácie a simulácie a znemožňoval použitie v reálnom čase na zariadeniach s menej výkonným hardvérom. Tieto dôvody viedli k vykonaniu optimalizácie v nasledovných oblastiach:

- Redukcia počtu homogénnych transformačných matíc.

  Oddelené matice translácií a rotácií reprezentujúce elementárnu transláciu alebo rotáciu boli spojené do jednej homogénnej transformačnej matice popisujúcej translácie vo všetkých troch osiach trojrozmerného priestoru, čo predstavuje článok (z angl. *link*) a následnú elementárnu rotáciu vo zvolenej osi, čo predstavuje rotačný kĺb (z angl. *joint*). Aplikovanie tejto zmeny viedlo k zníženiu výpočtovej náročnosti a zároveň značky XML súboru popisujúceho štruktúru kinematického modelu robota museli byť prispôsobené tejto zmene. Nasledujúce úryvky textov demonštrujú zmenu z pôvodného značenia kinematického modelu na nové značenie.

```
# Povodne XML znacenie kinematickeho modelu
<KinematicPart Axis="TX" Value="121.0"/>
<KinematicPart Axis="TY" Value="-71.0"/>
<KinematicPart Axis="RZ" Value="0.2617"/>
# Nove XML znacenie kinematickeho modelu
<Link x="121.0" y="-71.0" z="0.0">
    <Joint axis="z" angle="0.2617"/>
</Link>
```

Ak by boli načítané pôvodné XML značky z úryvku do dátových štruktúr zdrojovým kódom z bakalárskej práce, potom by transformácia koncového bodu bola reprezentovaná troma maticami. Nové značenie a upravený zdrojový kód redukuje tieto tri transformačné matice na jednu.

- Náhrada pôvodných pomalých funkcií z knižnice *numpy* ich ekvivalentom knižnice *math*.

  Matematické funkcie knižnice *numpy* sú orientované na rýchle spracovanie vektorov alebo n-rozmerných matíc. Ich aplikovanie na jednohodnotové premenné dátového typu *int* alebo *float* je výpočtovo neefektívne. Rotačná zložka v transformačných maticiach si vyžaduje jednohodnotové výpočty pomocou funkcií sínus a kosínus, ktoré sú efektívnejšie implementované práve v knižnici *math*.

- Aktualizovanie hodnôt existujúcich transformačných matíc namiesto generovania nových matíc.

  Aktualizácia elementárnych premenných dátového typu *int* alebo *float* vrámci alokovanej inštancie matice v pamäti počítača je pre Python interpreter oveľa efektívnejšie ako alokovanie pamäte pri vytváraní kompletne novej inštancie matice dátového typu *numpy.ndarray* s aktualizovanými hodnotami.

- Redukcia množstva riadkov kódu a volaní metód.

  Prostredníctvom funkcie *dis* vstavaneého modulu *dis* jazyka Python boli vypísané *CPython bytecode* inštrukcie metód jednotlivých tried. Interpretovaný jazyk Python je typický tým, že čím má zdrojový kód programu menej riadkov, tým je menej bytecode-ových inštrukcií a zároveň rýchlosť tohto programu je vyššia. No nie vždy platí táto zásada a existujú výnimky, kedy môže menej riadkov kódu spomaliť chod programu. Preto boli inštrukcie všetkých metód analyzované touto funkciou a na základe ich množstva a typu boli pozmeňo-

vané alebo redukované časti pôvodného zdrojového kódu pri zachovaní jeho funkcionality.

Pre porovnanie výpočtovej rýchlosti novej optimalizovanej a starej neoptimalizovanej verzie modulu kinematiky bol vytvorený testovací skript *Benchmark.py*. *Benchmark.py* inicializuje kinematický model robota Scolopendra z XML súboru a vypočíta uhly natočenia kĺbov vo vzdialených relatívnych pozíciach voči končatinám pomocou inverznej kinematiky. Meranie času je spustené od okamžiku začatia výpočtu inverznej kinematiky robota Scolopendra a zastavené je hneď po jeho skončení. Program bol testovaný na jednodoskovom mikropočítači Raspberry Pi 3B+ taktovanom na 1.4GHz. Nasledovný text zobrazuje konzolový výstup s výpočtovým časom oboch modulov v milisekundách.

```
# Pred optimalizaciou          Po optimalizacii
Elapsed time: 63.8662 ms       Elapsed time: 8.0085 ms
```

Z konzolového výstupu je možné vidieť, že optimalizovaná verzia zdrojového kódu kinematiky je približne osemnásobne rýchlejšia ako predošlá a dokáže fungovať v reálnom čase aj na jednodoskovom mikropočítači Raspberry Pi 3B+[20].

4.3 Modul Visualization.py

Tretí modul je zameraný na vizualizáciu modelu robota. Využíva prvky knižnice Open3D, ktorá umožňuje načítavať súbory formátov 3D objektov, vykresliť ich do okna vizualizácie a manipulovať s nimi. Konštrukcia kĺbového robota sa skladá z veľkého množstva 3D objektov vzájomne viazaných kĺbmi alebo pevnými spojmi, ktoré sú definované štruktúrou jeho kinematického modelu. Túto funkcionalitu však knižnica Open3D nemá zahrnutú. Modul obsahuje triedu *VirtualModel*, ktorá dokáže na základe kinematického modelu vytvoreného z dátových štruktúr modulu kinematiky prepojiť transformácie objektov s transformáciami kĺbov vizualizovaného robota a pohybovať nimi. Transformácie z kinematického modelu sú aplikovateľné, iba ak je zhodný počet jeho transformačných matíc s počtom skupín sieťovín virtuálneho

modelu. Metódou *Transform* triedy *VirtualModel* je možné aplikovať transformácie získané metódou *GetTransformations* z inštancie triedy *KinematicModel*. Následným zavolaním metódy *Update* je vykonaná aktualizácia virtuálneho modelu v okne vizualizácie. Virtuálny model robota vo vizualizácii je načítaný z XML súboru s vlastnými značením, ktoré popisujú:

- lokálnu cestu k adresáru s súbormi sieťovín vrámci softvérového balíka,
- lokálne translácie a rotácie skupiny sieťovín,
- lokálne translácie a rotácie sieťovín,
- farbu sieťovín,
- názvy súborov sieťovín,
- prednastavenú viditeľnosť sieťovín v okne vizualizácie.

Okrem triedy *VirtualModel* obshauje modul ďalšie dve triedy *VirtualCamera* a *Visualization*. Trieda *VirtualCamera* predstavuje virtuálnu kameru, ktorá sprostredkováva užívateľovi pohľad na virtuálny model. Parametre kamery je možné načítať zo súboru formátu JSON. Súbor obsahuje nastavenia parametrov extrinsickej matice pohľadu kamery, veľkosť uhla záberu (FoV – z angl. *Field of View*) a priblíženie (z angl. zoom). *Visualization* je triedou vizualizačného okna a zároveň aj úložným kontajnerom pre všetky zobrazované 3D objekty. Nastavenie užívateľského pohľadu vo vizualizácii je možné pridaním inštancie objektu triedy *VirtualCamera* s načítanými parametrami. Načítaním súboru formátu JSON je možné nastaviť parametre okna vizualizácie, ktorými sú jeho veľkosť v pixeloch, absolútna pozícia vrámci obrazovky monitora v pixeloch, farbu pozadia a názov okna.

4.4 Modul Package.py

Modul *Package.py* je jadrom celej knižnice, ktorý obsahuje iba jednu triedu *Package*. Funkciou inštancie triedy *Package* je generovanie dátových štruktúr z modulov kniž-

nice *Core* inicializovaných obsahom súborov softvérového balíka. Týmto spôsobom sú všetky dátové štruktúry knižnice spravované z jedného centrálneho miesta. Zníži sa tak množstvo importovaných modulov v zdrojovom kóde vytváranej aplikácie, ktorá bude inštanciu implementovať.

Statickou metódou *Create* je možné vygenerovať nový inicializovaný softvérový balík pre vybraného kĺbového robota. Každý novo-vytvorený softvérový balík má jasne definované základné usporiadanie a pomenovanie adresárov. Programátor si však môže rozširovať balík o svoje vlastné adresáre podľa preferencií. Adresár *Apps* je pripravený pre spustiteľné aplikácie. Adresár *Files* obsahuje subadresáre pomenované podľa príslušných štandardných formátov súborov, pre ktoré sú pripravené. *Utilities* je určený pre súbory zdrojových kódov modulov s dodatočnou funkcionalitou, nevyhnutnou pre fungovanie špecifického robota napr. hardvérové ovládače, špeciálne matematické moduly a pod.

Vytvorený balík okrem adresárov obsahuje v koreni súbor hlavného modulu v tvare *NázovRobotaPackage.py* so zdrojovým kódom v jazyku Python. Modul obsahuje jednu definovanú triedu s rovnakým názvom ako je názov modulu. Dedenie všetkých vlastnosti a metód triedy *Package* umožňuje generovať inštancie všetkých tried z knižnice *Core*. Okrem toho je pripravená na generovanie inštancií tried z modulov adresára *Utilities* vrámci softvérového balíka. Funkcionalitu generovania si však programátor musí doprogramovať.

Posledným prvkom softvérového balíka je inicializačný súbor formátu JSON s názvom *Settings.json*, ktorý je taktiež v koreni balíka. Súbor je vždy načítaný pri vytvorení novej inštancie triedy z hlavného modulu v ľubovoľnej aplikácii. *Settings.json* obsahuje absolútnu cestu k softvérovému balíku a zároveň je aj odkazovým súborom na ostatné inicializačné súbory balíka uložené v subadresároch adresára *Files*.

5 Softvérový balík Scolopendra

Prostredníctvom knižnice *Core* bol vytvorený softvérový balík pre robota Scolopendra. Balík má usporiadanú štruktúru do formátu, ktorý je popísaný v sekcii 4.4.

Adresár *Utilities* obsahuje súbory modulov so zdrojovými kódmi špeciálne prispôsobenými na vývoj aplikácií pre hexapoda Scolopendra. Všetky moduly adresára sú funkčne nezávislé okrem *Movements.py*, ktorý implementuje dátové štruktúry modulu *Trajectories.py*. Zdrojové kódy oboch modulov sú rovnako ako modul kinematiky základného balíka prevzaté z mojej bakalárskej práce. *Trajectories.py* obsahuje triedy translačného a rotačného generátora jednej parabolickej trajektórie. *Movements.py* obsahuje triedy translačných a rotačných generátorov parabolických trajektórií a zároveň zabezpečuje radenie koncových bodov končatín do celkového pohybu podľa vybraného typu [20]. Oba moduly boli optimalizované tak, aby dosiahli za čo najkratší čas výpočet súradníc pre koncové body končatín robota v nasledujúcej iterácii. Optimalizácia zdrojových kódov bola vykonaná podobnými postupmi ako v prípade modulu *Kinematics.py* knižnice *Core*.

VirtualMovements.py je doplňujúcim modulom pre generátory pohybov, ktorý umožňuje pridať generované trajektórie do vizualizácie. Modul implementuje prvky knižnice Open3D a tým je zachovaná kompatibilita s modulom knižnice *Visualization.py*.

ServoDrivers.py je modul z bakalárskej práce s minoritnými zmenami v zdrojovom kóde. Je určený na riadenie dvoch PWM ovládačov servo motorov PCA9685 prostredníctvom jednodoskového mikropočítača Raspberry Pi[20].

Modul *JointsAdjustments.py* nahradil triedu *CSVPreprocessor* z bakalárskej práce. Jeho funkcionalita je identická t.j. načítavanie uhlov natočenia kĺbov robota zo súboru. Rozdielmi medzi nimi sú však spôsoby načítavania.

1. Trieda *CSVPreprocessor* načítava obsah súborov formátovaných do štandardu *CSV*. Modul *JointsAdjustments.py* číta obsah JSON súboru.

2. Jeden súbor *.csv* obsahoval vždy jednu sekvenciu natočenia kĺbov. Preto mu-

sela mať každá jedna sekvencia vytvorený separátny súbor. Formátovanie JSON podporuje vnáranie objektov, čo dovoľuje všetky sekvencie uložiť do jedného súboru pod unikátnym kľúčom.

Posledný modul je *Multicast.py*. Dopĺňa funkcionalitu modulu *Communication* základnej knižnice *Core*. Trieda *Multicast* má definovaný jeden pevne definovaný serverový TCP socket a variabilný počet klientských TCP socketov, ktorými sa pripojí na príslušné spustené serverové aplikácie. Pripojená klientská aplikácia na multicast dokáže rozposielať pakety všetkým serverovým aplikáciám. Multicast príjme klientsky paket zo socketu a odošle jeho obsah na všetky otvorené klientské sockety. Klientskej aplikácii je vrátená kópia obsahu posledného prijatého paketu s odpoveďou. Ak ani jeden zo serverov nie je pripojený multicast je ukončený. Podmienkou správneho fungovania multicastu je schopnosť serverových aplikácií odpovedať na prijatý paket a zároveň mať definovaný koncový bod s odpoveďou k tomuto paketu.

Ďalší adresár s názvom *Apps* obsahuje aplikácie vytvorené z modulov balíka v jazyku Python. Vrámci adresára sú aplikácie rozdelené podľa viazanosti na operačný systém. Aplikácie, ktoré nie sú viazané na operačný systém, ale sú v koreni adresára, sú spustiteľné na ľubovoľnom operačnom systéme s nainštalovaným Python interpreterom. Okrem aplikácií naprogramovaných v jazyku Python je v adresári aplikácia grafického užívateľského rozhrania naprogramovaná v jazyku C#, ktorá je podrobnejšie popísaná v kapitole 5.3.

Adresár *Files* obsahuje konfiguračné súbory, na základe ktorých sa inicializujú všetky inštancie tried vo vytvorených aplikáciách. Súbory sú utriedené v subadresároch podľa formátu, v akom je ich obsah definovaný. Adresár *JSON* obsahuje konfiguračné súbory formátované v štandarde JSON. Tento štandard je určený pre inicializáciu väčšiny dátových štruktúr vrámci softvérového balíka. Adresár *XML* obsahuje konfiguračné súbory formátované v štandarde XML. Formát XML je využívaný iba pre zápis štruktúry kinematického modelu, virtuálneho modelu vizualizácie a iných vizualizovaných 3D objektov. Adresár *STL* obsahuje súbory sieťovín 3D objektov a simplifikovaných komponentov navrhnutého modelu robota v binárnom

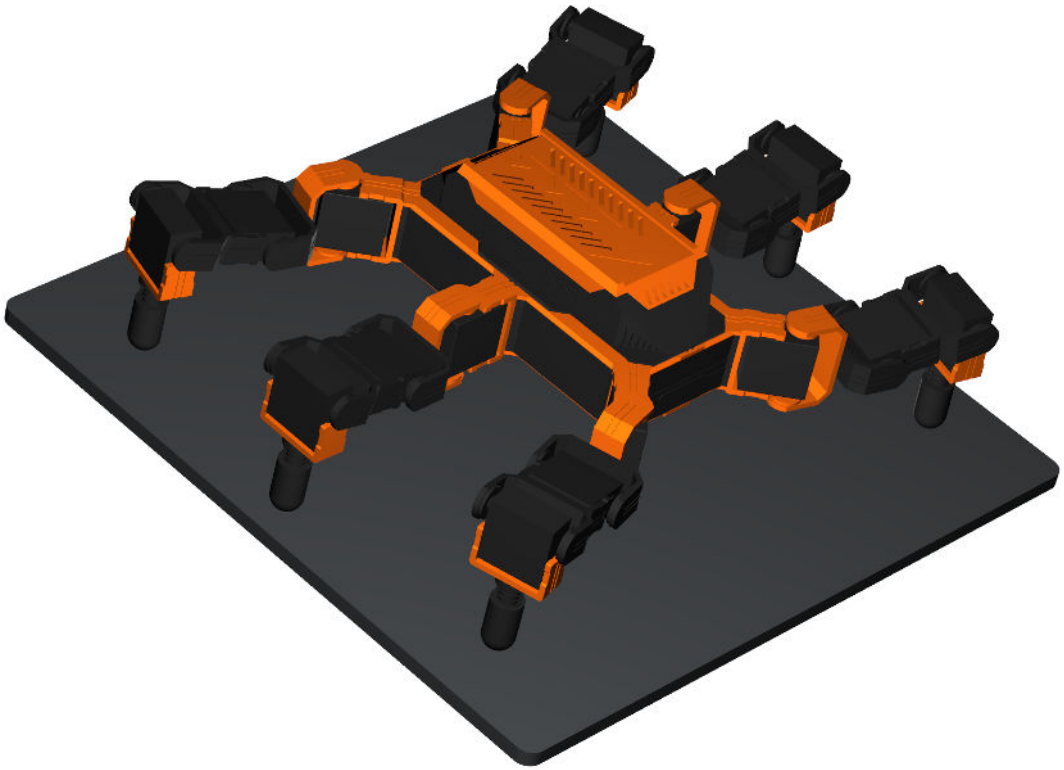
zápise formátu *.stl*.

5.1 Virtuálny model robota v prostredí vizualizácie

Serverová aplikácia vizualizácie so zdrojovým kódom v súbore *VisualizationServer.py* modelu robota Scolopendra je vytvorená z dátových štruktúr z modulov knižnice *Core* inicializovaných súbormi softvérového balíka. Dodatočná funkcionalita je sprostredkovaná špeciálnymi modulmi softvérového balíka adresára *Utilities*. Všetky dátové štruktúry tvoriace výslednú aplikáciu sú načítané z inštancie triedy *ScolopendraPackage*. Tá ich vytvorí z prelinkovaných súborov softvérového balíka. Štruktúra kinematického modelu robota Scolopendra je načítaná zo súboru *KinematicModel.xml*. Pôvodný XML súbor z bakalárskej práce popisoval kinematický model 46 značkami, čo sa prejavilo v programe ako 46 transformačných matíc. Nové značenie a fungovanie modulu redukovalo množstvo značiek a matíc na 24.

Virtuálny model robota je zložený z 32 odľahčených komponentov navrhnutého modelu z bakalárskej práce a 25 virtuálnych koordinačných rámov. Odľahčenie komponentov virtuálneho modelu bolo vykonané zjednotením pôvodných komponentov na základe lokálnej viazaností medzi dvoma kĺbmi a spoločnej farby. Ďalej boli odstránené upevňovacie diery a členitosť niektorých komponentov bola znížená, aby sa zmenšil počet vrcholov a stien sieťovín. V konečnom dôsledku prinieslo odľahčenie komponentov zníženie výpočtovej náročnosti na zariadenie, na ktorom je spustená aplikácia vizualizácie. Súbor *VirtualModel.xml* popisuje usporiadanie jednotlivých komponentov virtuálneho modelu robota Scolopendra vo virtuálnom priestore vizualizácie. Okrem virtuálneho modelu robota Scolopendra sú vo vizualizácii pridané aj doplnkové grafické prvky:

1. podložka,
2. trajektórie,
3. emulovaný sledovaný objekt záujmu.



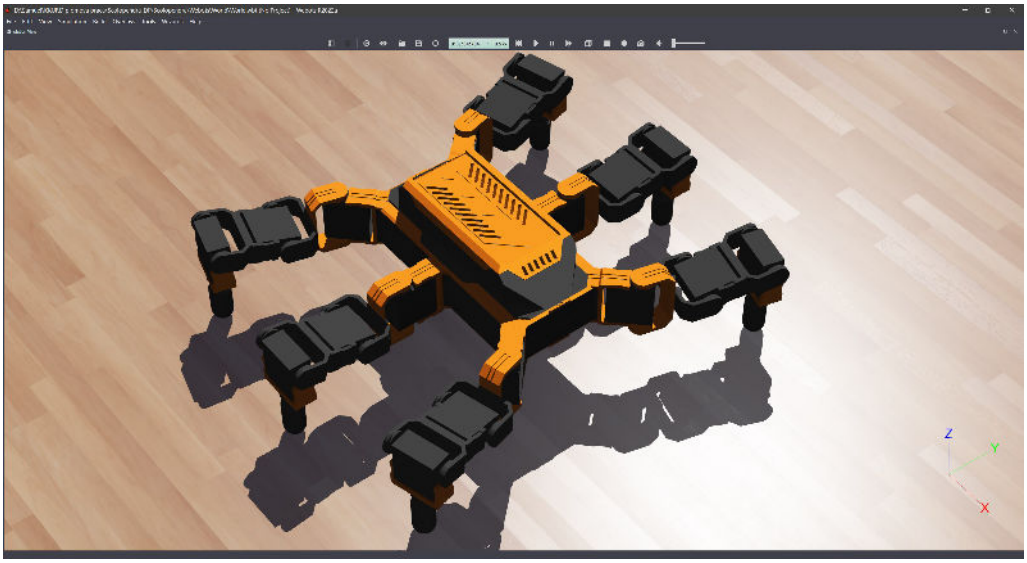
Obrázok 5 – 1 Vizualizácia modelu robota Scolopendra

Inštancia triedy *JsonServer* je v aplikácii inicializovaná JSON súborom s IP adresou 127.0.0.1 a portom 27003. Má definovaných 19 aktívnych koncových bodov komunikácie, prostredníctvom ktorých dokáže pripojená klientská aplikácia zasahovať do vizualizácie a vykonávať v nej vizuálne zmeny.

5.2 Virtuálny model robota v prostredí simulátora Webots

Vizualizácia robota Scolopendra modulmi knižnice *Core* plní iba funkciu zobrazovania stavu virtuálneho modelu pevne viazaného v jednej polohe, ktorý však nepodlieha mechanicko-fyzikálnym vlastnostiam prostredia ako sú gravitácia, zotrvačnosť, kolízie s okolitými objektami a pod. Aplikácia Webots poskytuje možnosti modelovania vlastného robota a následnej simulácie jeho správania v prostredí s príslušným spusteným ovládačom. Virtuálny robot Scolopendra v prostredí simulácie bol navrhnutý prevažne z interných prvkov obsiahnutých v aplikácii simulátora. Ako kolízne

objekty robota definujúce fyziku v simulácii boli použité primitívne tvary – kvádre a kapsuly, pretože komplexnejšie tvary s väčším počtom vrcholov by nadmerne zaťažovali výpočtovú jednotku počítača alebo by mohlo dôjsť k nesprávnemu fungovaniu fyzikálneho enginu. Pohon robota zabezpečuje 18 virtuálnych rotačných motorov. Každý motor má priradený pozičný senzor, aby spĺňal funkcionalitu fyzického servo motora so spätnou väzbou. Na dizajn komponentov modelu v simulácii však boli použité sieťoviny z rovnakého adresára ako pri vytváraní virtuálneho modelu vo vizualizácii, aby softvérový balík neobsahoval redundantné súbory. Obrázok (6 – 4) zobrazuje robota Scolopendra v prostredí simulátora Webots.



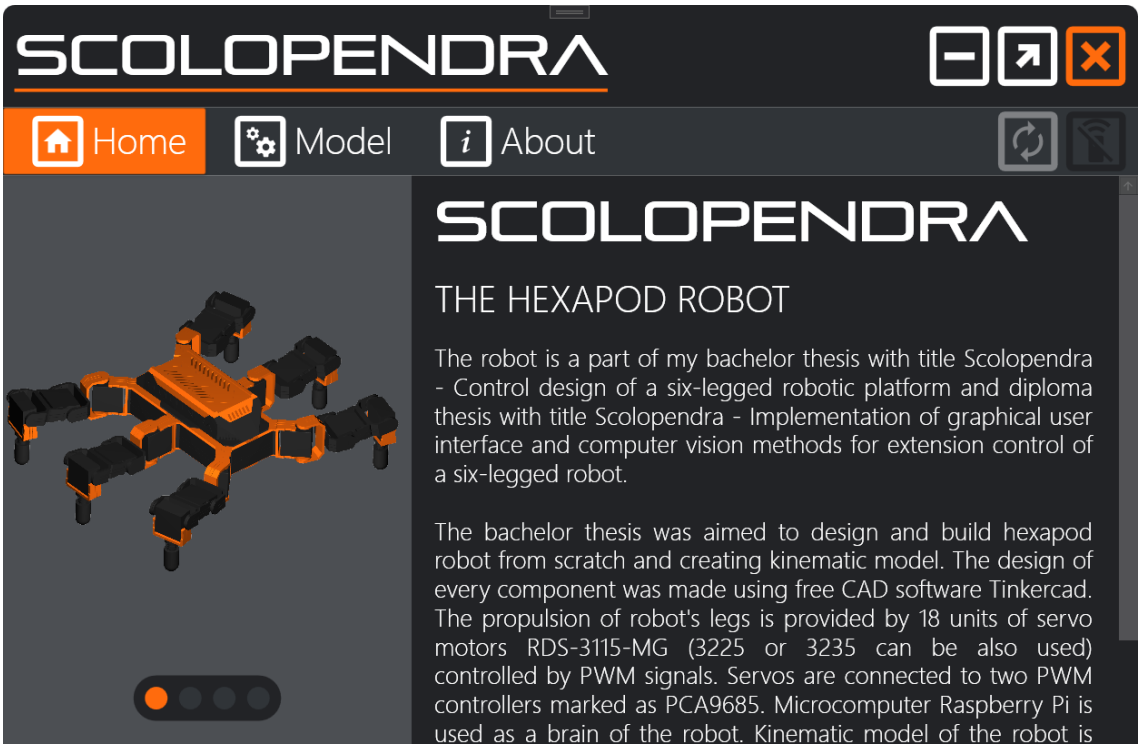
Obrázok 5 – 2 Virtuálny model robota Scolopendra v prostredí simulátora Webots

Pre programovanie modelu v simulátore bol vytvorený interný doplnkový softvérový balík s názvom adresára *Webots*. Balík *Webots* je iba doplnkom softvérového balíka *Scolopendra*, čo znamená, že nie je nevyhnutný pre jeho fungovanie. Obsahuje modul ovládača všetkých 18 virtuálnych servo motorov *VirtualServoDriver* a modul *WebotsScolopendraPackage*, ktorý dedí všetky vlastnosti od triedy *ScolopendraPackage*. Dedenie od tejto triedy zabezpečuje trojstupňové dedenie. Okrem prístupu k jej metódam a premenným je možný prístup aj ku všetkým metódam a premenným triedy *Package* z knižnice *Core*.

Serverová aplikácia ovládača robota v simulátore Webots so zdrojovým kódom v súbore *WebotsServer.py* je súčasťou doplnkového balíka. Štruktúra kinematického modelu robota Scolopendra je načítaná z rovnakého súboru *KinematicModel.xml* ako u modelu vizualizácie. Komunikácia je sprostredkovaná inštanciou triedy *JsonServer*, ktorá je inicializovaná JSON súborom s IP adresou 127.0.0.1 a portom 27002. Má definovaných 16 aktívnych koncových bodov komunikácie, prostredníctvom ktorých dokáže pripojená klientská aplikácia ovládať virtuálny model.

5.3 Aplikácia Scolopendra UI

Klientská aplikácia Scolopendra UI alebo Scolopendra User Interface (obrázok 5 – 3) je aplikáciou grafického užívateľského rozhrania pre ovládanie robota Scolopendra.



Obrázok 5 – 3 Pohľad na spustenú aplikáciu grafického rozhrania Scolopendra UI

Aplikácia obsahuje 7 rôznych funkcionalít, ktorými užívateľ dokáže ovládať modely robotov:

1. *Origin Transformation* – ovládanie ťažiska robota
2. *Forward Kinematics* – ovládanie natočenia kĺbov končatín robota
3. *Inverse Kinematics* – ovládanie pozícií koncových bodov končatín robota
4. *Camera Emulator* – emulácia kamery
5. *Vzdialené ovládanie* – ovládanie pohybu robota v priestore
6. *Ovládanie pohybov* – nastavovanie parametrov pohybu
7. *Viditeľnosť sieťovín* – nastavovanie viditeľnosti sieťovín (dostupné iba pre virtuálny model vo vizualizácii)

Komplexita týchto funkcionalít je natoľko rozsiahla, že zabezpečiť užívateľsky prijateľné ovládanie prostredníctvom CLI by bolo programátorsky nemožné. Zdrojové kódy aplikácie sú napísané v jazyku C#. Na grafické prevedenie aplikácie bol použitý deklaratívny značkovací jazyk XAML softvérového rámca WPF. Dizajn aplikácie je obohatený o obsah knižnice Material Design Themes. Kód celej aplikácie Scolopendra UI bol rozdelený na triedy, ktoré sú súčasťou:

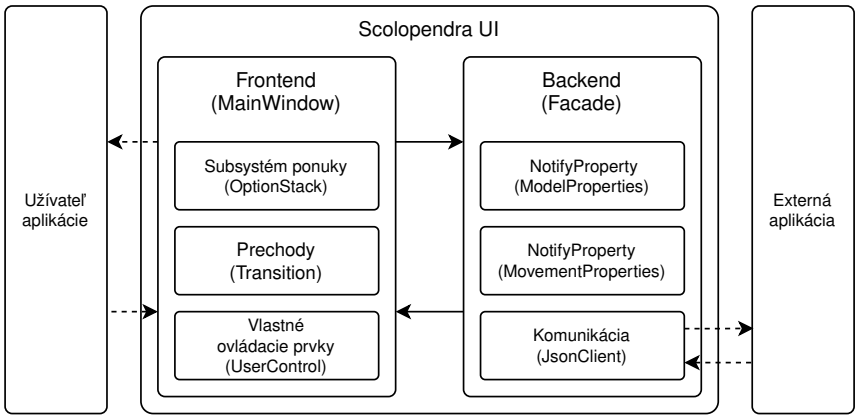
- Grafického prevedenia – frontend. Všetky triedy, s ktorými prichádza užívateľ do kontaktu:
  1. Subsystém ponuky – *OptionStack*
  2. Prechody – *Transitions*
  3. Vlastné ovládacie prvky – *UserControls*

  Tieto triedy nevykonávajú žiadnu aplikačnú logiku nevyhnutnú pre dosiahnutie užívateľom požadovaného stavu.
- Aplikačnej logiky – backend. Všetky inštancie, ktoré užívateľ nastavuje prostredníctvom grafických prvkov:

1. NotifyProperty – *ModelProperties*, *MovementProperties*
2. Komunikácia – *JsonClient*

Vykonávajú inštrukcie prijaté z frontendu, aby dosiahli užívateľom požadovaný stav. Aplikujú zmeny v grafických elementoch, ktoré nastali pri užívateľskom zásahu do rozhrania.

Obrázok (5 – 4) zobrazuje schému spôsobu uloženia tried v backende a frontende aplikácie Scolopendra UI.



Obrázok 5 – 4 Bloková schéma vnútorného usporiadania aplikácie Scolopendra UI

5.3.1 Subsystém ponuky

Prvým krokom pri tvorbe rozhrania je návrh hierarchického subsystému prepínania medzi obrazovkami, ktorý slúži ako navigácia medzi panelmi. Scolopendra UI využíva len jedno okno *MainWindow* pre zobrazovanie obrazoviek. Koncept jedného okna a prepínania panelov v rámci neho dodáva oveľa prijateľnejší užívateľský zážitok z angl. *User eXperience* (UX) ako vytváranie nových okien s vlastnou funkcionalitou. Pre užívateľa by to mohlo pôsobiť mätúco a neprehľadne. Všetky obrazovky sú definované v tomto okne ako panely. Panely sú navzájom prekryté a nastavené tak, aby ich užívateľ nevidel. Funkcia prepínania panelov je implementovaná dátovou štruktúrou zásobníka z angl. *stack*. Podstatnou vlastnosťou zásobníka je princíp

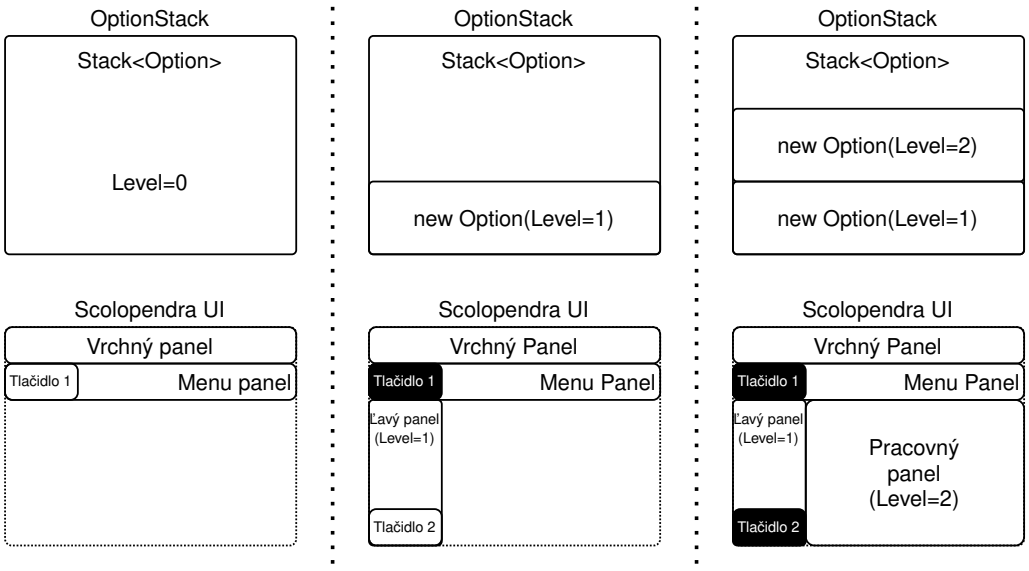
jeho fungovania pri ukladaní elementov nazývaný posledný dnu, prvý von z angl. *Last In, First Out* (LIFO). Kontajnérová trieda *Stack<T>* jazyka C# definuje 3 základné metódy:

1. *Peek* – prístup k posledne pridanému elementu,
2. *Push* – pridanie nového elementu na vrch,
3. *Pop* – odstránenie posledne pridaného elementu.

Kliknutím na tlačidlo je užívateľovi zobrazený panel viazaný na príslušné tlačidlo. Previazanosť tlačidla *SenderObject* a panelu *UserControl* zabezpečuje trieda *Option*. Panely majú rôznu úroveň uloženia – *Level*, ktorá je definovaná pri vytváraní inštancie triedy *Option*. Po otvorení panelu rovnakej úrovne je starý panel uvedený do neviditeľného stavu a nahradený novým zvoleným zviditeľneným panelom. Pri voľbe panelu o úroveň nižšie sa všetky panely až po danú úroveň skryjú a zviditeľní sa len zvolený panel. Všetky vytvorené inštancie triedy *Option* sú ukladané do dátovej štruktúry zásobníka *Stack<Option>*, ktorá je komponentom triedy *OptionStack*. Princíp hierarchického prepínania medzi panelmi demonštruje obrázok (5 – 5). Grafické rozhranie aplikácie Scolopendra UI má 2 úrovne vnárania. Prvé vnorenie obsahuje 3 voľby a spúšťa sa pri kliknutí na jedno z tlačidiel z vrchného menu. Voľba *Home* má definované iba jedno vnorenie. Druhé vnorenie možno vykonať iba pri voľbe *Model*, ktoré obsahuje až 8 volieb alebo pri voľbe *About* s dvoma voľbami vnárania.

5.3.2 Prechody

Softvérový rámec WPF poskytuje vytváranie animácií rôzneho typu, no ich nasadenie vo väčšej kvantite zanecháva veľké množstvo redundantného kódu, ktorý je možné zredukovať. Prechody sú sada tried naprogramovaných na jednoduché spúšťanie animácii bez písania nadbytočne sa opakujúceho kódu. Základným prvkom prechodov je abstraktná trieda *Transition*. Všetky vlastnosti triedy *Transition* dedia tri podtriedy:



Obrázok 5 – 5 Schéma prepínania panelov

1. *ColorTransition* – wrapper trieda farebného prechodu z jednej farby do druhej,
2. *DoubleTransition* – wrapper trieda jednoparametrového spojitého prechodu,
3. *ThicknessTransition* – wrapper trieda štvorparametrového spojitého prechodu odsadenia.

Všetky tri podtriedy implementujú verejný interface *ITransition* s jednou metódou *void Run(ITransition)*. Metóda *Run* abstraktnej triedy *Transition* je definovaná ako virtuálna. Polymorfizmom je možné dosiahnuť to, že všetky triedy dediace vlastnosti abstraktnej triedy môžu virtuálnu metódu *Run* prepisovať a definovať tak vlastnú funkcionalitu.

V aplikácii Scolopendra UI bolo spolu použitých 45 prechodov z toho 22 farebných prechodov, 22 jednoparametrových spojitých prechodov a 1 štvorparametrový prechod odsadenia.

5.3.3 Vlastné ovládacie prvky

WPF dovoľuje vytvárať vlastné ovládacie prvky so špecifickou funkcionalitou a grafickým prevedením (Kapitola 3.3). Každý programátorom vytvorený ovládací prvok

dedí vlastnosti triedy *UserControl*. Vrámci aplikácie Scolopendra UI sú implementované vlastné grafické prvky v dvoch podobách ako:

1. navigačné panely – užívateľský zásah do týchto prvkov mení aktuálny ,
2. ovládacie panely pre riadenie pohybu modelov robota.

Vzhľadom na ich veľký počet a komplexnosť celej aplikácie sú analyzované iba vybrané prvky riadenia robota:

- **ControlCameraEmulator**

  Ovládací prvok *ControlCameraEmulator* je špeciálne určený na emulovanie fungovania kamery pripojenej na výpočtovú jednotku so spusteným programom aplikácie vyhľadávania objektu zájmu. Dizajnové prevedenie kamerového emulátora v grafickom rozhraní aplikácie Scolopendra UI zobrazuje obrázok (5 – 6).

Obrázok 5 – 6 Dizajnové prevedenie kamerového emulátora

Softvérovo emulovaná kamera s vyhľadávaním nie je súčasťou fyzického robota Scolopendra, ale je možné ju nahradiť hardvérom. Pozícia objektu zájmu na

emulovanej obrazovej rovine kamery smerovanej v doprednom smere pohybu robota je reprezentovaná oranžovým terčom a karteziánskymi súradnicami x a y. Súradnica z manipuluje s jeho veľkosťou, čo emuluje približovanie a vzdiaľovanie objektu záujmu.

Presunutím kurzora myši na oranžový terč, následným stlačením, držaním ľavého tlačidla a pohybom po ohraničenom paneli ho dokáže užívateľ presúvať a meniť tak jeho súradnice x a y vrámci pracovného priestoru. Rolovaním kolieska na myši vpred alebo vzad sa aplikuje zmena súradnice z, ktorá sa na obrazovke prejaví zmenou veľkosti oranžového terča. Stlačením kolieska alebo kliknutím na ikonu oka v oranžovom obdĺžniku je zviditeľnený alebo zneviditeľní objekt záujmu vo virtuálnom prostredí vizualizácie.

Veľkosť pracovného priestoru terča je 600 jednotiek do šírky pričom hodnoty sú z intervalu $\langle -300.0; 300.0 \rangle$, 400 jednotiek do výšky z intervalu $\langle -200.0; 200.0 \rangle$ a 50 jednotiek do hĺbky z intervalu $\langle -25.0; 25.0 \rangle$. Pomer strán x a y je štandardizovaný na 4:3. Tým je uľahčený transformačný prepočet pri použití fyzickej kamery s pomerom strán 4:3 pokiaľ by bola upevnená na fyzickom robotovi Scolopendra.

- **ControlVirtualJoystick**

  Ovládací prvok *ControlVirtualJoystick* je softvérovým prevedením hardvérového joysticku s príslušnou funkcionalitou. Dizajnové prevedenie virtuálneho joysticku v grafickom rozhraní aplikácie Scolopendra UI zobrazuje obrázok (5 – 7). Presunutím kurzora myši do oblasti vnútorného kruhu následným stlačením ľavého tlačidla a jeho posúvaním je možné ovládať joystick. Pustením ľavého tlačidla sa joystick automaticky vráti do pôvodného stavu.

  Súradnice z joysticku sú normalizované na jednotkovú kružnicu a môžu byť načítané v:

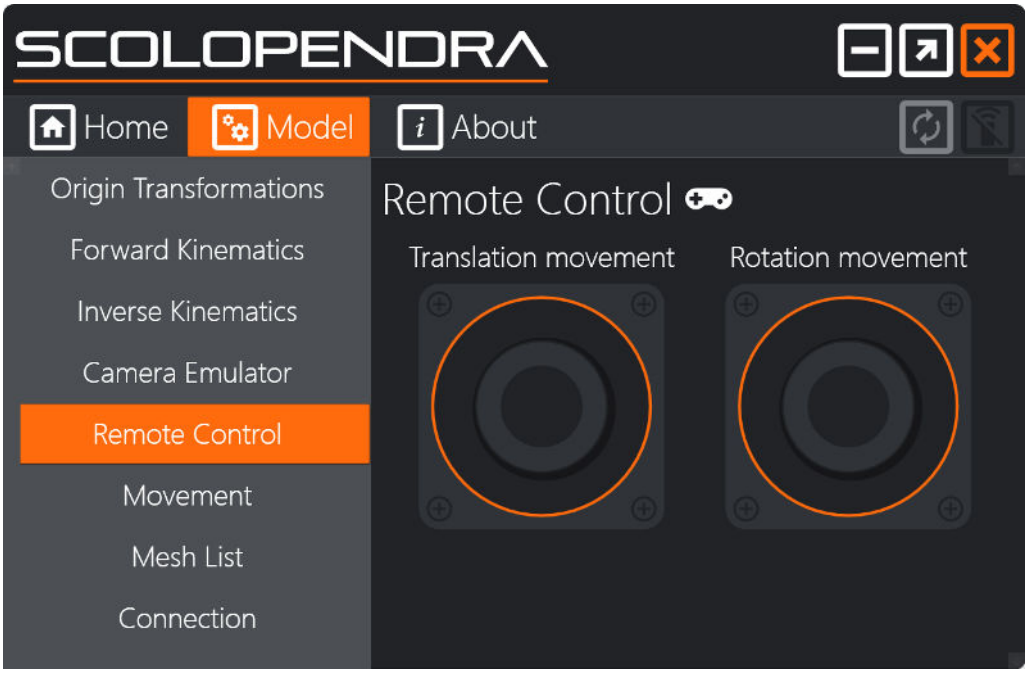
  - karteziánskom tvare ako dvojica hodnôt (x, y), kde $x \in \langle -1.0; 1.0 \rangle$ je

horizontálna zložka a $y \in \langle -1.0; 1.0 \rangle$ je vertikálna zložka,

- polárnom tvare ako dvojica hodnôt (ρ, ϕ), kde $\rho \in \langle 0.0; 1.0 \rangle$ je dĺžka vektora a $\phi \in (-\pi, \pi\rangle$ je uhol otočenia voči vertikálnej osi otáčania.

V grafickom rozhraní Scolopendra UI boli implementované dve inštancie ovládacieho prvku *VirtualJoystick*.

1. Prvým joystickom je ovládaný translačný pohyb robota s využitím polárnej reprezentácie súradníc. Dĺžka vektora ρ nastavuje rýchlosť zmeny parametra translačného pohybu a uhol ϕ určuje smer pohybu.
2. Druhým joystickom je ovládaný rotačný pohyb robota na mieste s využitím karteziánskej reprezentácie súradníc. Tu je využívaná iba horizontálna zložka x, ktorej znamienko určuje smer otáčania a veľkosť nastavuje rýchlosť zmeny parametra rotačného pohybu. Vertikálna zložka y je zanedbaná.



Obrázok 5 – 7 Dizajnové prevedenie virtuálneho joysticku

- **ControlTranslationMovement**

  Ovládací prvok *ControlTranslationMovement* je ovládacím panelom, ktorý umožňuje užívateľovi meniť parametre generátora translačného pohybu:

  – viditeľnosť trajektórií (*visibility*),

  – vzor pohybu (*pattern*),

  – výška (*height*),

  – šírka (*width*),

  – uhol (*angle*).

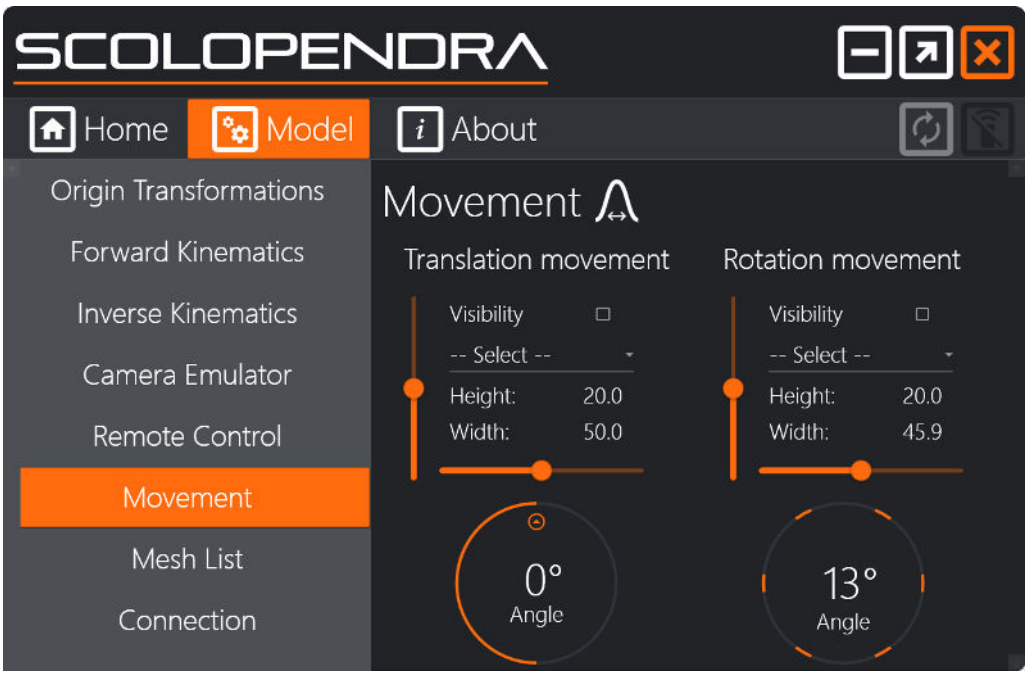
  Dizajnové prevedenie ovládacieho panelu translačného pohybu v grafickom rozhraní aplikácie Scolopendra UI zobrazuje obrázok (5 – 8). Viditeľnosť trajektórie je ovládaná zaškrtávacím poľom – *CheckBox*. Vzor pohybu je prepojený s grafickým prvkom rozbaľovacieho poľa – ComboBox, ktorého grafické prevedenie je rozšírené o dizajn z knižnice *Material Design Themes*. Vzor pohybu má 3 možnosti na výber – *Tripod*, *Quadruped* a *Crawl*. Horizontálnym posuvníkom – *Slider*, sa mení šírka trajektórie z rozsahu $\langle 10.0; 90.0\rangle$ milimetrov. Vertikálnym posuvníkom sa mení výška trajektórie z rozsahu $\langle 10.0; 30.0\rangle$ milimetrov. Posuvníky sú taktiež obohatené o dizajn knižnice podobne ako rozbaľovacie pole pre vzor pohybu . Posledný grafický prvok slúži na nastavovanie uhla z rozsahu $(-180.0°; 180.0°\rangle$. Na dizajnové prevedenie bol použitý ukazovateľ postupu – *ProgressBar*. Statický zdroj knižnice *Material Design Themes* transformuje jeho výzor z predvoleného lineárneho dizajnu na kruhový. Okrem vylepšenej grafickej stránky je rozšírený o funkcionalitu reagovania na užívateľský zásah. Presunutím kurzora myši do oblasti ikony šípky v kružnici spodného grafického prvku, následným stlačením ľavého tlačidla a jeho posúvaním je možné nastavovať smer translačného pohybu.

- **ControlRotationMovement** Ovládací prvok *ControlRotationMovement* je ovládacím panelom, ktorý umožňuje užívateľovi meniť parametre generátora

rotačného pohybu:

- viditeľnosť trajektórií (*visibility*),
- vzor pohybu (*pattern*),
- výška (*height*),
- uhol (*angle*).

Dizajnové prevedenie ovládacieho panelu rotačného pohybu v grafickom rozhraní aplikácie Scolopendra UI zobrazuje obrázok (5 – 8). Dizajn a ovládanie je takmer rovnaké ako v prípade panela translačného pohybu. Rozdiel je v absencii nastavovania parametra šírky. Tá je vyjadrená uhlom otočenia robota počas jednej periódy otočenia z rozsahu $\langle 0.0°; 25.0°\rangle$ okolo osi z. Zobrazovaná hodnota šírky je len orientačná, pretože závisí od polomeru, ktorého veľkosť je definovaná vzdialenosťou bodu pohybu od osi otáčania.



Obrázok 5 – 8 Dizajnové prevedenie ovládacích panelov pohybov

5.3.4 NotifyProperty

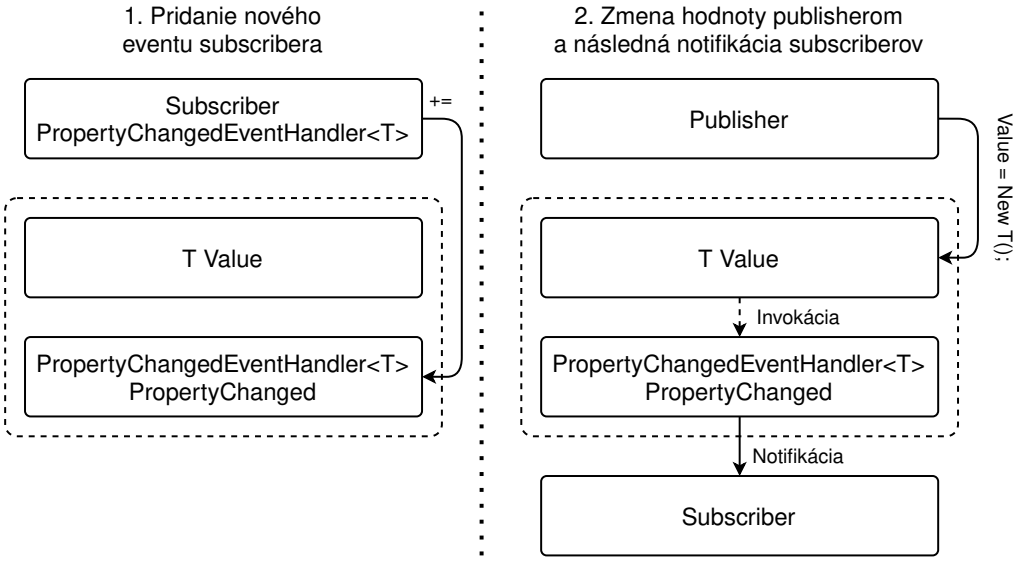
Grafické elementy rozhrania je potrebné automaticky aktualizovať z backendu pri vykonaní užívateľského akčného zásahu vo frontende. *NotifyProperty<T>* je trieda naprogramovaná k tomuto účelu.

Sufix *<T>* je označovaný ako generikum z angl. *generic* alebo generický dátový typ. Generické dátové typy predstavujú koncept typovej parametrizácie, ktorá je aplikovateľná pri deklarácii tried, štruktúr, delegátov, interface-ov, metód alebo tried. Umožňujú programátorovi definovať jeden alebo viac parametrov podľa požiadaviek jeho kódu. Dátový typ dosadený za typový parameter je automaticky rozpoznaný kompilátorom v procese kompilácie programu do strojového kódu.

Trieda *NotifyProperty<T>* enkapsuluje jeden generický dátový typ <T> do inštancie ako privátnu premennú *value* a zároveň disponuje verejnou vlastnosťou udalosti s názvom *PropertyChanged* dátového typu *PropertyChangedEventHandler<T>*. *NotifyProperty<T>* využíva návrhový vzor pozorovateľa *observer*. Premenná *value* má definované prístupové práva čítania – *getter* a zápisu – *setter*. Setter premennej *value* je okrem funkcie nastavovania rozšírený o schopnosť invokácie udalosti *PropertyChanged*. Premenná *PropertyChanged* je úložné miesto externých udalostí od entít – *subscriber-ov*, ktoré sa majú vykonať pri zmene parametra *value*. Nastavovanie *value* vykonáva externá entita označovaná ako *publisher*. Po nastavení sú invokované všetky akumulované udalosti a dochádza ku notifikácii subscriberov. Podrobný popis fungovania triedy zobrazuje obrázok (5 – 9).

Výhodou tohto prístupu je jednoduchosť aplikovania na ľubovoľný dátový typ, vykonanie zmien z jedného centrálneho miesta a neobmedzenosť množstva externých udalostí subscriberov.

Na druhú stranu používanie *NotifyProperty* v kóde programu má aj jeden nedostatok a tým je možný vznik nekonečnej rekurzie pokiaľ je programátor nedôsledný a v konečnom dôsledku program skončí výnimkou *StackOverflowException*. Rekurzia nastáva, ak jeden zo subscriberov je zároveň aj publisherom. Tento problém je však



Obrázok 5 – 9 Diagram fungovania triedy *NotifyProperty<T>*

riešiteľný a to temporálnym zabránením prístupu k nastavovaniu premennej *value* prostredníctvom settera publisherovi zodpovednému za jeho vznik.

Pri programovaní aplikácie Scolopendra UI bolo celkovo použitých 13 inštancií triedy *NotifyProperty<T>* na všetky parametre, ktoré sú odosielané a prijímané zo serverovej aplikácie.

V backende aplikácie boli použité 2 inštancie tried, ktoré implementujú *NotifyProperty<T>*:

1. *ModelProperties*

   - trieda všetkých *NotifyProperty<T>* parametrov popisujúcich aktuálny stav virtuálneho modelu robota, ktoré sú načítavané zo servera – natočenie kĺbov robota, pozícia koncových bodov končatín, pozícia stredu robota a viditeľnosť sieťovín (využívané iba pri spustenej vizualizácii).

2. *MovementProperties*

   - trieda všetkých *NotifyProperty<T>* parametrov popisujúcich aktuálny stav generátorov translačného a rotačného pohybu, ktoré sú načítavané zo

servera – výška trajektórie, šírka trajektórie (iba pri transačnom pohybe), smerový uhol, viditeľnosť (využívané iba pri spustenej vizualizácii).

5.3.5 Komunikácia

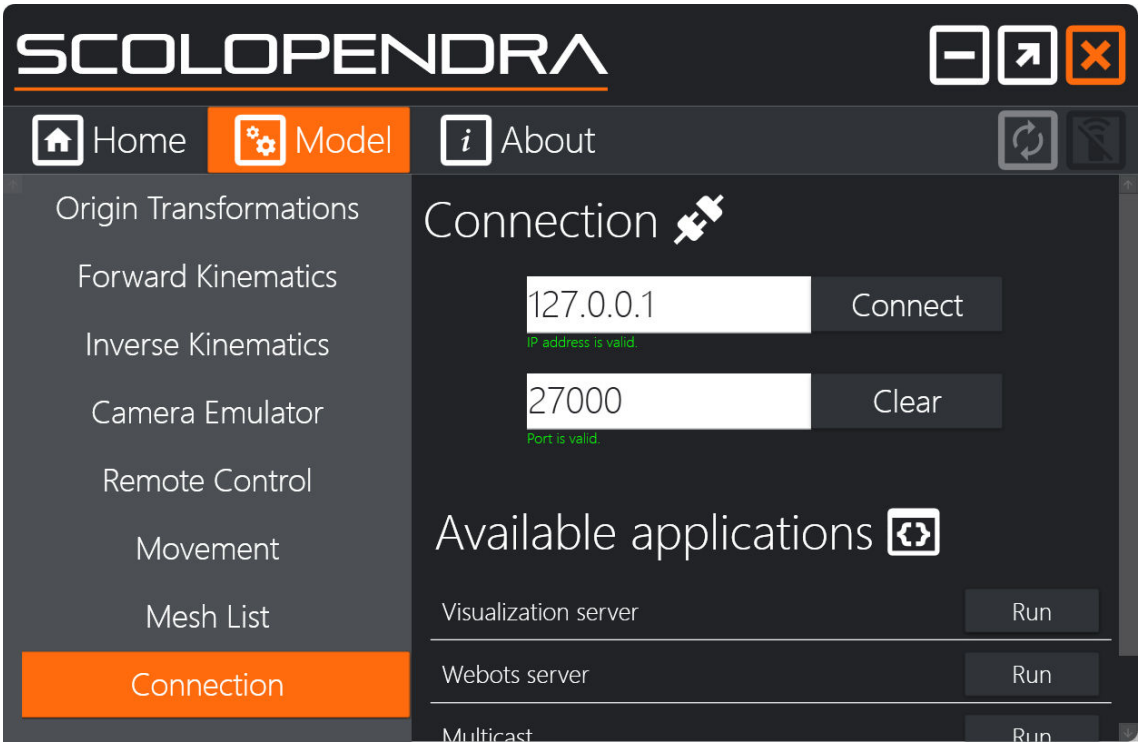
Komunikáciu Scolopendra UI so serverovou aplikáciou zabezpečuje backendová trieda *JsonClient* s pomocnou triedou *Request*. *JsonClient* má rovnakú funkcionalitu ako trieda *JsonClient* z knižnice *Core* so zdrojovým kódom napísanom v jazyku Python. Inštancia tejto triedy v spustenej klienstkej aplikácii je schopná komunikovať s jednou inštanciou triedy *JsonServer*, ktorá je súčasťou knižnice *Core*. Interným komponentom triedy je streamový TCP socket sieťového rozhrania. Zavolaním jednej z preťažených metód *Connect* alebo *ToggleConnection* (pokiaľ nie je klient pripojený) sa spúšťa asynchrónne pripojenie streamového socketu na server. Zavolaním metódy *Disconnect* alebo *ToggleConnection* (pokiaľ je klient pripojený) sa ukončuje komunikačné prepojenie. Počas volania týchto funkcií môže byť volaná jedna z troch vlastnosti (z angl. *property*) dátového typu akcie – *Action*:

1. *OnDismissed* – akcia spustená pri neúspešnom nadviazaní komunikácie,
2. *OnConnected* – akcia spustená pri úspešnom nadviazaní komunikácie,
3. *OnDisconnected* – akcia spustená pri ukončení komunikácie.

Akcie umožňujú agregovať delegátov neparametrických externých *void* metód, ktoré sú spustené naraz z jedného centrálneho miesta pri invokácii. Pomocná trieda *Request* umožňuje jednoduché generovanie obsahu paketu vo formáte JSON. Pri jej programovaní bol využitý návrhový vzor statickej metódy továrne – *static factory method*. Názov statickej metódy definuje druhý stupeň vnárania. Prvý vstupný parameter dátového typu *string* definuje prvý stupeň vnárania *Topic*. Posledný parameter dátového typu *object* je objekt určený na serializáciu ako hodnota kľúča *Content*. Vytvorený objekt dátového typu *Request* je vstupným parametrom do metódy

Request<T> triedy *JsonClient*. Podobne ako pri programovaní triedy *NotifyProperty<T>* aj tu bol využitý generický dátový typ definujúci dátový typ výstupneho parametra funkcie.

Grafické rozhranie Scolopendra UI implementuje jednu inštanciu triedy *JsonClient*. Nastavenie parametrov pripojenia – IP adresa a port servera, je možné vyplnením textových polí v paneli pripojenia – *Connection*, ktorý zobrazuje obrázok 5 – 10.



Obrázok 5 – 10 Panel nastavenia komunikácie

Po stlačení tlačidla *Connect* môžu nastať dve situácie:

1. klientská aplikácia sa úspešne pripojí na spustený proces servera, čo indikuje zelená farba ikony v pravom hornom rohu, ktorá je volaná pri akcii *OnConnected*,

2. klientská aplikácia nedokázala nadviazať spojenie so serverovou aplikáciou, pretože užívateľ poskytol nesprávne parametre alebo aplikácia nebola spus-

tená, čo indikuje bliknutie ikony v pravom hornom rohu načerveno invokovaná akciou *OnDismissed*.

Pripojením na server je možné dostupnými grafickými ovládacími prvkami vykonávať užívateľský akčný zásah a odosielať tak requesty na 19 rôznych koncových bodov, ktoré vykonávajú čítanie, zápis alebo aktualizáciu virtuálneho alebo fyzického modelu robota Scolopendra.

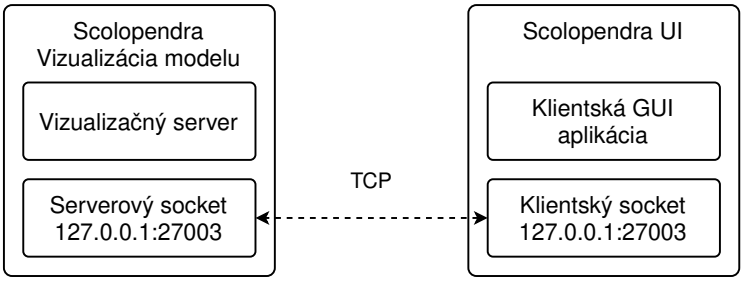
6 Kombinácie prepajáteľných aplikácií

Spustené aplikácie softvérového balíka robota Scolopendra je možné prepájať v rôznych kombináciách podľa potrieb užívateľa alebo podľa výkonu hadvéru. Každá aplikácia ľubovoľného modelu, či už virtuálneho alebo fyzického robota Scolopendra, má svoj vlastný nezávislý výpočet kinematického modelu a trajektórií. Serverové aplikácie získavajú vstupné dáta na základe užívateľského vstupu od pripojenej klientskej aplikácie a spätne odosielajú pozmenené hodnoty, ktoré sú aktualizované v grafických prvkoch rozhrania. Rôzným prepojením vytvorených aplikácií softvérového balíka robota Scolopendra je možné dosiahnuť 10 možných kombinácií prepojení:

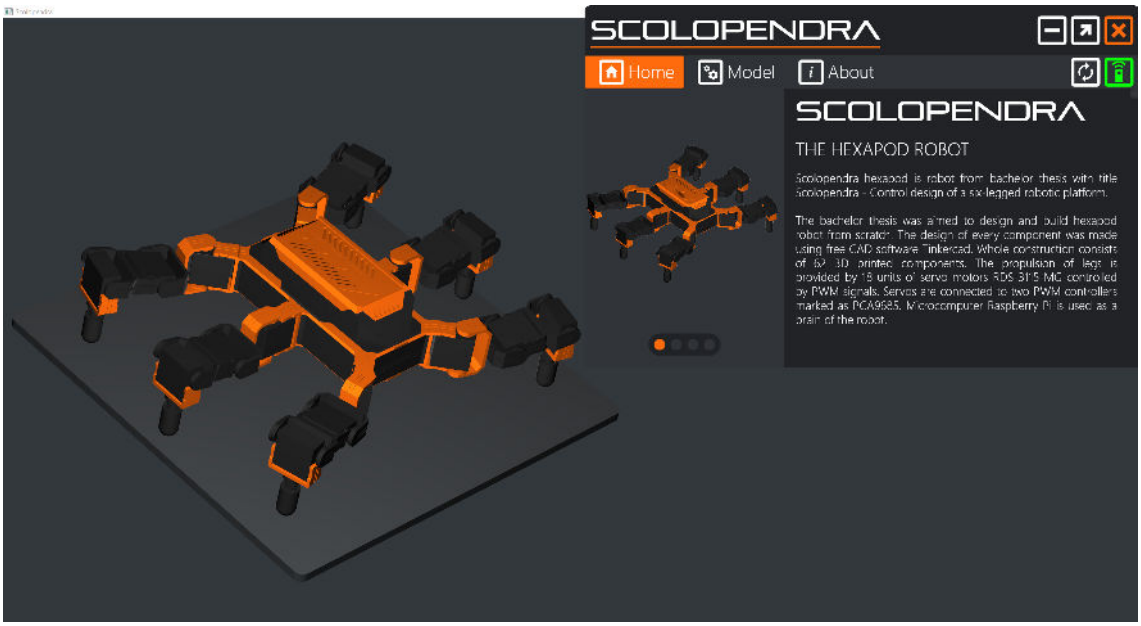
1. Scolopendra UI a vizualizácia Open3D bez multicastu,
2. Scolopendra UI a simulácia Webots bez multicastu,
3. Scolopendra UI a fyzický robot bez multicastu,
4. Scolopendra UI, vizualizácia Open3D a multicast.
5. Scolopendra UI, simulácia Webots a multicast.
6. Scolopendra UI, fyzický robot a multicast.
7. Scolopendra UI, vizualizácia Open3D, simulácia Webots a multicast.
8. Scolopendra UI, vizualizácia Open3D, fyzický robot a multicast.
9. Scolopendra UI, simulácia Webots, fyzický robot a multicast.
10. Scolopendra UI, vizualizácia Open3D, simulácia Webots, fyzický robot a multicast.

6.1 Scolopendra UI a vizualizácia Open3D

Prvá kombinácia prepojenia je medzi klientskou aplikáciou grafického užívateľského rozhrania a serverovou aplikáciou Open3D vizualizácie virtuálneho modelu robota Scolopendra. Vizualizácia je spustiteľná iba na operačnom systéme Windows. Pre pripojenie Scolopendra UI je potrebné zadať IP adresu 127.0.0.1 a port 27003 do príslušných textových polí a stlačiť tlačidlo *Connect*. Obrázok (6 – 1) zobrazuje komunikačnú architektúru prepojenia týchto dvoch aplikácií. Obrázok (6 – 2) zobrazuje stav prepojenia týchto dvoch aplikácií.



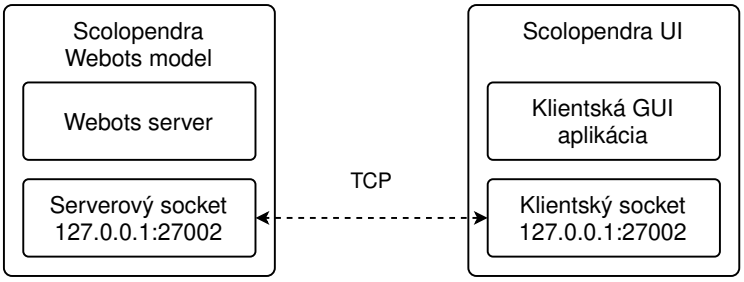
Obrázok 6 – 1 Schéma architektúry prepojenia GUI a vizualizácie



Obrázok 6 – 2 Pohľad na prepojené aplikácie GUI a vizualizácie

6.2 Scolopendra UI a simulácia Webots

Druhá kombinácia prepojenia je medzi klientskou aplikáciou grafického užívateľského rozhrania a serverovou aplikáciou virtuálneho modelu robota Scolopendra v prostredí simulátora Webots. Pre pripojenie Scolopendra UI je potrebné zadať IP adresu 127.0.0.1 a port 27002 do príslušných textových polí a stlačiť tlačidlo *Connect*. Obrázok (6 – 3) zobrazuje komunikačnú architektúru prepojenia týchto dvoch aplikácií. Obrázok (6 – 4) zobrazuje stav prepojenia týchto dvoch aplikácií.

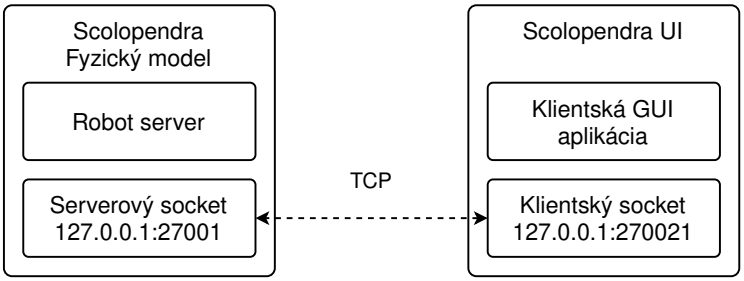


Obrázok 6 – 3 Schéma architektúry prepojenia GUI a Webots simulácie

Obrázok 6 – 4 Pohľad na prepojené aplikácie GUI a simulátora Webots

6.3 Scolopendra UI a fyzický robot

Tretia kombinácia prepojenia je medzi klientskou aplikáciou grafického užívateľského rozhrania a serverovou aplikáciou fyzického modelu robota Scolopendra. Pre pripojenie Scolopendra UI je potrebné zadať IP adresu 127.0.0.1 a port 27001 do príslušných textových polí a stlačiť tlačidlo *Connect*. Obrázok (6 – 5) zobrazuje komunikačnú architektúru prepojenia týchto dvoch aplikácií. Obrázok (6 – 6) zobrazuje stav prepojenia fyzického robota a grafického rozhrania.

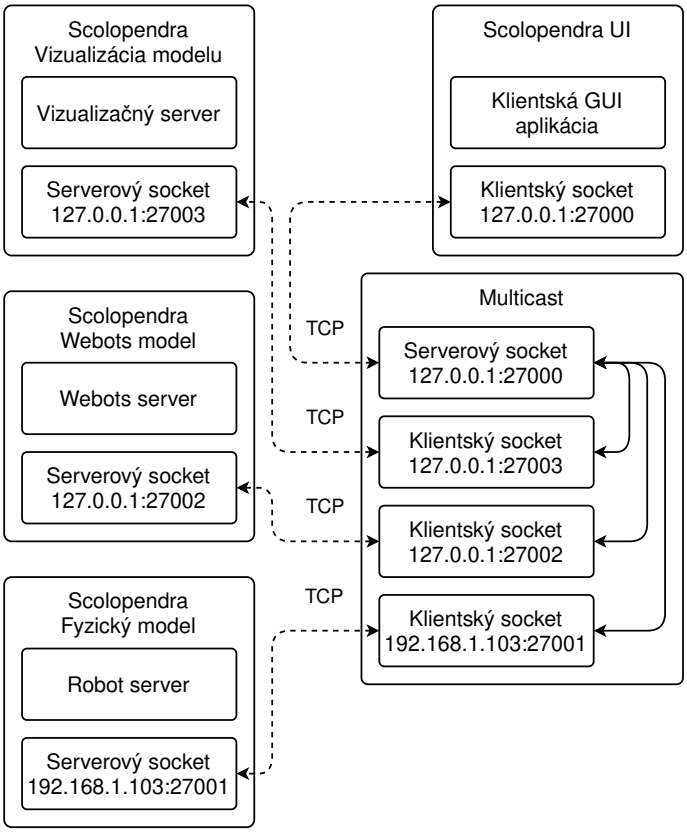


Obrázok 6 – 5 Schéma architektúry prepojenia GUI a fyzického robota

Obrázok 6 – 6 Pohľad na prepojenú aplikáciu GUI s fyzickým robotom

6.4 Scolopendra UI a multicast

Zvyšných sedem kombinácií prepojenia je medzi klientskou aplikáciou grafického užívateľského rozhrania a serverovými aplikáciami prostredníctvom aplikácie multicastu. Pre pripojenie Scolopendra UI k multicastu je potrebné zadať IP adresu 127.0.0.1 a port 27000 do príslušných textových polí a stlačiť tlačidlo *Connect*. Aplikácia multicastingu sa správa ako rozdeľovač prijatého TCP paketu od klienta. Prijatý paket je rozposlaný všetkým serverom, na ktorý sa multicasting dokázal pripojiť počas inicializácie. Multicasting odosiela klientovi spätne paket od posledného pripojeného servera v poradí. Pre fungovanie tejto architektúry je nevyhnutné mať spustenú aspoň jednu serverovú aplikáciu. Obrázok (6 – 7) zobrazuje komunikačnú architektúru prepojenia aplikácií pri využití multicastingu.



Obrázok 6 – 7 Schéma architektúry prepojenia GUI a multicast aplikácie

6.5 Testovanie funkcionalít

V procese testovania bolo zisťované, či naprogramované funkcionality všetkých serverových aplikácií sú ovládateľné prostredníctvom grafického rozhrania v súlade s hardvérovými, softvérovými a bezpečnostnými obmedzeniami.

Testovanými funkcionalitami boli:

- Ovládanie ťažiska
- Priama kinematika
- Inverzná kinematika
- Kamerový emulátor
- Diaľkové ovládanie
- Ovládanie pohybov
- Viditeľnosť sieťovín

Serverová aplikácia vizualizácie má dostupných všetkých sedem funkcionalít bez akéhokoľvek obmedzenia.

V serverovej aplikácii simulácie Webots nie je možné ovládať viditeľnosť sieťovín. Túto funkcionalitu nebolo možné naprogramovať, keďže simulátor Webots nedisponuje možnosťou manipulovať s viditeľnosťou vložených 3D objektov.

V serverovej aplikácii fyzického modelu funkcionalita nastavovania viditeľnosti tiež nie je implementovaná, keďže sa jedná iba čisto o softvérovú záležitosť. Navyše má aplikácia fyzického modelu deaktivované funkcionality zápisu do priamej a inverznej kinematiky. Dôvodom deaktivácie boli bezpečnostné obmedzenia. Počas testovania na virtuálnych modeloch bolo zistené, že pri týchto dvoch funkcionalitách môže nastať stav, kedy sa kĺby končatín robota dostanú do fyzicky nemožného stavu alebo vznikne medzi končatinami vzájomná kolízia. Dôsledkom chybného stavu by

bolo poškodenie alebo zničenie elektronických komponentov fyzického modelu robota. Pri priamej kinematike je možné dosiahnuť tento stav, ak užívateľ nastaví v grafickom rozhraní uhol natočenia kĺbu končatiny mimo fyzicky povolený rozsah. V prípade inverznej kinematiky môže nastať chybný stav, ak aproximačná metóda inverzným jakobianom nebude schopná nájsť riešenie inverznej kinematiky alebo dosiahne stav konvergencie v inej konfigurácii, čo môže spôsobiť, že sa kĺby končatín dostanú mimo povolený rozsah. Výsledkom testovania je tabuľka 6 – 1, ktorá zobrazuje dostupnosť ovládania funkcionality z grafického rozhrania.

Funkcionalita	**Vizualizačný server**	**Simulačný server**	**Fyzický server**
Ovládanie ťažiska	Dostupná	Dostupná	Dostupná
Priama kinematika	Dostupná	Dostupná	Nedostupná
Inverzná kinematika	Dostupná	Dostupná	Nedostupná
Kamerový emulátor	Dostupný	Dostupný	Dostupný
Diaľkové ovládanie	Dostupné	Dostupné	Dostupné
Ovládanie pohybov	Dostupné	Dostupné	Dostupné
Viditeľnosť sieťovín	Dostupná	Nedostupná	Nedostupná

Tabuľka 6 – 1 Dostupnosť funkcionalít serverových aplikácií

6.6 Experiment merania výpočtového času

Cieľom experimentu bolo meranie času jedného výpočtu hlavnej slučky programov všetkých serverových aplikácií. Čas bol meraný pri aplikovaní translačného pohybu na model robota prostredníctvom virtuálneho joysticku z grafického rozhrania. Diaľkové ovládanie spúšťa generovanie trajektórií a ich aplikovanie na kinematický model robota. Virtuálne modely boli spúšťané na desktopovom počítači. Hardvérové parametre testovacieho desktopového zariadenia sú:

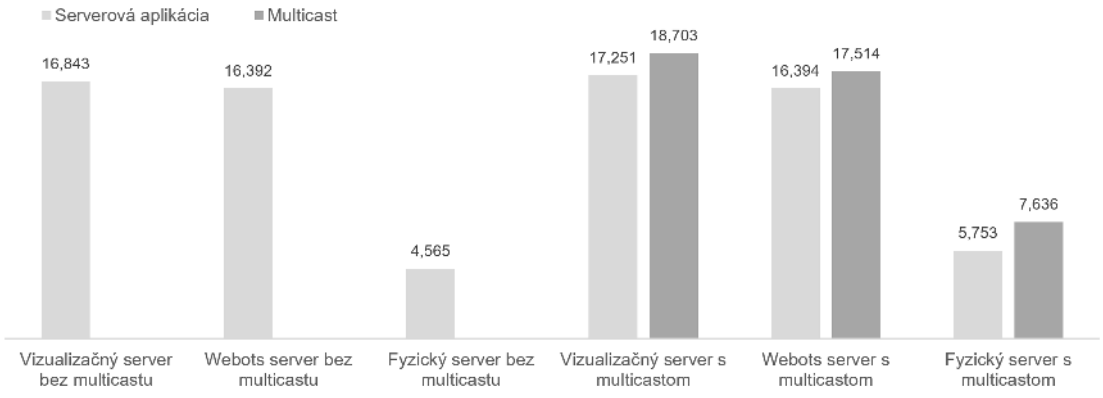
- Centrálny procesor – AMD Ryzen 7 5700X
- Grafický procesor – NVIDIA RTX 3070
- Operačná pamäť – 32GB DDR4 3600MHz

Fyzický model bol riadený jednodoskovým mikropočítačom Raspberry Pi 3B+. Riadiaca jednotka bola pripojená na počítač, ktorý mal spustenú funkcionalitu Wi-Fi prístupového bodu. Meraním času jedného výpočtu sa overovalo, či generovanie trajektórií a výpočet priamej a inverznej kinematiky nadmerne nezaťažuje výpočtové jednotky. Ďalej bol meraný aj výpočtový čas multicastu. Z nameraných hodnôt bolo možné zistiť činnosť aplikácií v reálnom čase a vplyv multicastu na výpočtový čas serverových aplikácií. Nadmerná záťaž výpočtovej jednotky by mala za následok pomalú odozvu na užívateľský vstup z grafického rozhrania a nízku obnovovaciu frekvenciu aktualizácie zobrazovaných virtuálnych modelov. Výstupom experimentu bolo desať meraní, ktoré boli rozdelené na dve skupiny a zapísané v dvoch tabuľkách (Tabuľka 6 – 2 a 6 – 3). Ku každej tabuľke bol vytvorený stĺpcový graf (Obrázok 6 – 8 a 6 – 9) pre lepšie porovnanie nameraných hodnôt. Prvú skupinu predstavovalo šesť meraní časov jednej spustenej serverovej aplikácie pripojenej na klientskú aplikáciu bez zapnutého multicastu a so zapnutým multicastom. Zvyšné štyri merania prebiehali pri zapnutom multicaste a viacerých spustených serverových aplikáciách. Tabuľky obsahovali spolu 22 priemerných hodnôt časov získaných rôznou kombináciou

spustených serverových aplikácií. Priemerný čas bol vypočítaný z 1000 nameraných vzoriek spustenej aplikácie v milisekundách.

Serverová aplikácia	Výpočtový čas aplikácie	Výpočtový čas multicastu
Visualizačný server bez multicastu	16.843 ms	-
Webots server bez multicastu	16.392 ms	-
Fyzický server bez multicastu	4.565 ms	-
Visualizačný server s multicastom	17.251 ms	18.703 ms
Webots server s multicastom	16.394 ms	17.514 ms
Fyzický server s multicastom	5.753 ms	7.636 ms

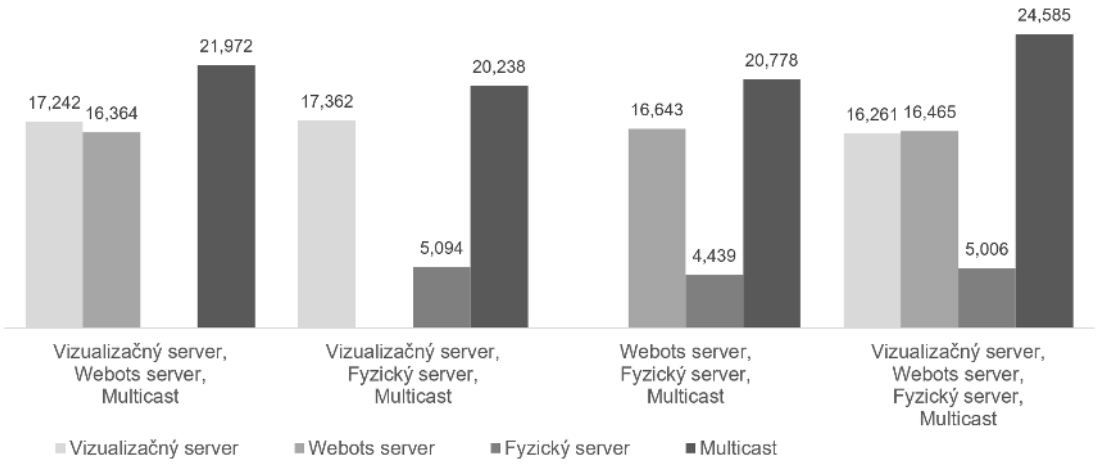
Tabuľka 6 – 2 Časy výpočtov jednej spustenej serverovej aplikácie



Obrázok 6 – 8 Graf časov výpočtov jednej spustenej serverovej aplikácie

Kombinácia serverových aplikácií	Vizualizačný server	Webots server	Fyzický server	Multicast
Webots server, Vizualizačný server, Multicast	17.242 ms	16.364 ms	-	21.972 ms
Fyzický server, Vizualizačný server, Multicast	17.362 ms	-	5.094 ms	20.238 ms
Fyzický server, Webots server, Multicast	-	16.643 ms	4.439 ms	20.778 ms
Webots server, Fyzický server, Vizualizačný server, Multicast	16.261 ms	16.465 ms	5.006 ms	24.585 ms

Tabuľka 6 – 3 Časy výpočtov kombinácií spustených serverových aplikácií



Obrázok 6 – 9 Graf časov výpočtov kombinácií spustených serverových aplikácií

Z prvej časti experimentu zameraného na jednu serverovú aplikáciu bolo zistené, že najnižší čas výpočtu má aplikácia fyzického modelu robota a najvyšší čas výpočtu má aplikácia vizualizácie. Desktopový počítač je oveľa výkonnejším zariadením ako jednodoskový mikropočítač Raspberry Pi 3B+, no jeho limitujúcim faktorom je neustále grafické vyobrazovanie virtuálneho modelu robota. Raspberry Pi vykonáva namiesto vyobrazovania modelu priame riadenie servo motorov, čo nemá až taký zásadný vplyv na výpočtovú záťaž procesora. Taktiež je možné si povšimnúť, že vizualizácia Open3D je oproti simulácii Webots výpočtovo náročnejšia. Tento fakt je daný tým, že aplikácia simulátora Webots je naprogramovaná v programovacom jazyku C++ a logika aplikácie vizualizácie s využitím knižnice Open3D je naprogramovaná v jazyku Python. Interpretovaný jazyk Python je mnohonásobne pomalší ako kompilovaný nízkoúrovňový jazyk C++, a tým je aj výsledná aplikácia pomalšia. Pre všetky tri aplikácie však platí, že výpočtový čas so spusteným multicastom bol o trochu vyšší ako bez neho. Dôvodom je dvojúrovňové odosielanie paketov z klientskej aplikácie do multicastu a následne z multicastu na serverovú aplikáciu.

Z druhej časti vykonaného experimentu merania zameraného na kombinácie serverových aplikácií vyplýva, že najviac zaťažujúcou kombináciou pre desktopové zariadenie je trojkombinácia všetkých aplikácií. Prejavilo sa to hlavne na multicaste, ktorý mal najvyšší čas výpočtu. Pri simultánnom ovládaní všetkých troch modelov z grafického rozhrania nebola registrovaná extrémne pomalá reakcia na užívateľský zásah v porovnaní s ostatnými kombináciami. Pri dostatočne výkonnom zariadení dokážu byť paralelne spustené všetky serverové aplikácie s grafickým rozhraním a reagovať na užívateľský vstup takmer okamžite. Pri menej výkonnom zariadení je možné znížiť záťaž tak, že sa zvolí dvojkombinácia alebo iba jedna spustená aplikácia.

7 Záver

Táto diplomová práca bola zameraná na softvérové rozšírenie hardvérovo zameranej bakalárskej práce. Teoretická analýza uviedla všetky oblasti, do ktorých zasahuje problematika práce, ich výhody a nevýhody. V prehľad technológií boli spracované podklady všetkých implementovaných technológií.

Praktické realizácia tvorila väčšiu časť diplomovej práce. Vytvorená knižnica *Core* v jazyku Python je vhodným základom pre prácu so softvérovými balíkmi mobilný kĺbových robotov alebo kĺbových robotov s pevne viazanou základňou. Knižnica zahrňuje optimalizovaný zdrojový kód modulu kinematiky z bakalárskej práce a taktiež modul vizualizácie využívajúci prvky knižnice *Open3D*, ktorými je možné vizualizovať virtuálny model kĺbového robota zo súboru formátu XML. Vytvorené komunikačné rozhranie s využitím TCP socketov a obsahu paketu formatovaného v štandarde JSON je funkčné, spoľahlivé a rýchle. Pre robota Scolopendra bol prostredníctvom knižnice *Core* vytvorený softvérový balík s názvom *Scolopendra* obsahujúci súbory zdrojových kódov a inicializačné súbory pre rýchle programovanie aplikácií. Súčasťou balíka sú vytvorené serverové aplikácie virtuálnych modelov robota Scolopendra v prostredí vizualizácie a simulácie. Vizualizácia bola realizovaná prvkami vlastného modulu vizualizácie postavenom na dátových štruktúrach knižnice Open3D. Pre simuláciu virtuálneho modelu robota bol použitý voľne dostupný robotický simulátor Webots. K programovaniu virtuálneho modelu robota v simulácii bol vytvorený doplnkový balík, ktorý obsahuje modul ovládača virtuálnych servo motorov a ovládač robota. Doplnenie projektu o virtuálne modely viedlo k mnohonásobnému uľahčeniu a urýchleniu vývoja a testovania softvéru. Balík obsahuje okrem iného aj plne funkčnú klientskú aplikáciu grafického užívateľského rozhrania s názvom Scolopendra UI. V závere práce boli otestované funkcionality všetkých troch serverových aplikácií a experimentálne merané časy rôznych kombinácií architektúr vzájomného prepojenia grafického rozhrania a serverových aplikácií.

Cieľom diplomovej práce však nebolo riešenie konkrétneho problému, ktorý by si

vyžadoval doplnenie modelu robota o dodatočný hardvér s požadovanou funkciou. Tá istá myšlienka platí pre softvér, ktorý nie je naprogramovaný pre dosiahnutie bližšie špecifikovaného riešenia. Celý projekt bol zameraný na vytvorenie extenzibilnej platformy založenej na šesťnohom robotovi pre ľudí s hardvérovými, ale aj softvérovými znalosťami, ktorí majú záujem pracovať na ďalšom rozšírení funkcionality tohto robota.

Literatúra

[1] Benny Raphael and Ian FC Smith. *Fundamentals of computer-aided engineering*. John wiley & sons, 2003.

[2] M.F. Robinette and R. Manseur. Robot-draw, an internet-based visualization tool for robotics education. *IEEE Transactions on Education*, 44(1):29–34, 2001. doi: 10.1109/13.912707.

[3] Vaibhav Gupta, Rajeevlochana G. Chittawadigi, and Subir Kumar Saha. Roboanalyzer: Robot visualization software for robot technicians. In *Proceedings of the Advances in Robotics*, AIR '17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450352949. doi: 10.1145/3132446.3134890. URL https://doi.org/10.1145/3132446.3134890.

[4] M Osman Tokhi and Abul KM Azad. *Flexible robot manipulators: modelling, simulation and control*, volume 68. Iet, 2008.

[5] C. Pepper, S. Balakirsky, and C. Scrapper. Robot simulation physics validation. In *Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems*, PerMIS '07, page 97–104, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595938541. doi: 10.1145/1660877.1660890. URL https://doi.org/10.1145/1660877.1660890.

[6] Bernard J Jansen. The graphical user interface. *ACM SIGCHI Bulletin*, 30(2): 22–26, 1998.

[7] Stefan Schaal. The sl simulation and real-time control software package. Technical report, Citeseer, 2009. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a729eb21ff7d95d2d247a220283c4490daf06411.

[8] Wisama Khalil, Aravindkumar Vijayalingam, Bogdan Khomutenko, Izzatbek Mukhanov, Philippe Lemoine, and Gaël Ecorchard. Opensymoro: An open-source software package for symbolic modelling of robots. In *2014 IEEE/ASME*

International Conference on Advanced Intelligent Mechatronics, pages 1206–1211, 2014. doi: 10.1109/AIM.2014.6878246.

[9] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[10] Webots. http://www.cyberbotics.com. URL `http://www.cyberbotics.com`. Open-source Mobile Robot Simulation Software.

[11] Matúš Smolko. Omicron – návrh swarm robota s rádiovou navigáciou. In *Technická univerzita v Košiciach, bakalárska práca*, page 37, 2021.

[12] Adele F. Scott and Changbin Yu. Cooperative multi-agent mapping and exploration in webots®. In *2009 4th International Conference on Autonomous Robots and Agents*, pages 56–61, 2009. doi: 10.1109/ICARA.2000.4803950.

[13] Chris Sells and Ian Griffiths. *Programming WPF: Building Windows UI with Windows Presentation Foundation*. Ö'Reilly Media, Inc.", 2007.

[14] Matthew MacDonald. Introducing wpf. In *Pro WPF in C# 2010: Windows Presentation Foundation in. NET 4.0*, pages 1–22. Springer, 2010.

[15] Lori A MacVittie. *XAML in a Nutshell*. Ö'Reilly Media, Inc.", 2006.

[16] Limi Kalita. Socket programming. *International Journal of Computer Science and Information Technologies*, 5(3):4802–4807, 2014.

[17] Norman Matloff. Tutorial on network programming with python. 2007.

[18] Abhijit A Sawant and B Meshram. Network programming in java using socket. *Google Scholar*, 2013.

[19] Wentao Shang, Yingdi Yu, Ralph Droms, and Lixia Zhang. Challenges in iot networking via tcp/ip architecture. *NDN Project*, 2016.

[20] Samuel Titko. Scolopendra – návrh riadenia šesťnohej robotickej platformy. In *Technická univerzita v Košiciach, bakalárska práca*, 2021.